1990

Guidance and control system for an Autonomous Underwater Vehicle

Cloutier, Michael John.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

GUIDANCE AND CONTROL SYSTEM
FOR AN AUTONOMOUS VEHICLE

by

Michael John Cloutier

June, 1990

Thesis Advisor: Yuh-jeng Lee
Second Reader: John M. Yurchak

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) Code 39	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Guidance and Control System for an Autonomous Vehicle

12. PERSONAL AUTHOR(S)
CLOUTIER, Michael J.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	June 1990	162

16. SUPPLEMENTARY NOTATION The views expre-sed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Autonomous Vehicle; Autonomous Underwater Vehicle; AUV; Local Path Planning; Real-Time Control; Cubic Spirals;

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Naval Postgraduate School (NPS) is currently involved in a long-term project to investigate and develop real-time control software, artificial intelligence, computer architecture and control systems theory as they pertain to U.S. Navy autonomous vehicle programs. In support of this goal, the NPS is currently designing and fabricating a testbed autonomous underwater vehicle.

This work describes the design, development, and testing of a Guidance Subsystem for this testbed vehicle which uses portions of cubic spirals as the desired path to follow between waypoints. In addition, data translation firmware and real-time software for the control surfaces and main motors is designed, implemented and tested.

The process of selecting and implementing an appropriate computer architecture in support of these goals is also discussed and detailed, along with the choice of associated

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Y. Lee/J. Yurchak	22b. TELEPHONE (Include Area Code) 408-646-2361/3390 22c. OFFICE SYMBOL CS/Le CS/Yu

DD Form 1473, JUN 86 *Previous editions are obsolete.*

computer hardware and real-time operating system software.

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution is unlimited.

Guidance and Control System
for an
Autonomous Underwater Vehicle

by

Michael John Cloutier
Lieutenant, United States Navy
B.S., Chapman College, 1982

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1990

Author:

Michael John Cloutier

Approved by:

Yuh-jeng Lee, Thesis Advisor

John M. Yurchak, Second Reader

Robert McGhee, Chairman
Department of Computer Science

ABSTRACT

The Naval Postgraduate School (NPS) is currently involved in a long-term project to investigate and develop real-time control software, artificial intelligence, computer architecture and control systems theory as they pertain to U.S. Navy autonomous vehicle programs. In support of this goal, the NPS is currently designing and fabricating a testbed autonomous underwater vehicle.

This work describes the design, development, and testing of a Guidance Subsystem for this testbed vessel which utilizes portions of cubic spirals as the desired path to follow between waypoints. In addition, data translation firmware and real-time control software for the control surfaces and main motors is designed, implemented and tested.

The process of selecting and implementing an appropriate computer architecture in support of these goals is also discussed and detailed, along with the choice of associated computer hardware and real-time operating system software.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

No undertaking as complex as a thesis can be completed without the combined effort, expertise, cooperation and understanding of many people. I am thankful to LCDR John M. Yurchak, a true 'C wizard', who gave me an appreciation of how effective properly written code can be. I am also indebted to Jim Bellingham for his cooperation and assistance with the Vehicle Control Computer software, without his help and advice I would still be lost. I am especially grateful to Yutaka Kanayama. His impressive knowledge of vehicle path-planning methods, combined with his extraordinary ability to clarify the most complex of problems, were instrumental in the development of the Guidance subsystem. His willingness to listen, and the inner peace which he radiates will remain with me always.

Lastly, I would like to acknowledge my wife, Chris, for her understanding, and the love and support which she provided while I labored on this difficult project. Without her sacrifices, this thesis would not have been possible.

I. INTRODUCTION

In the next decade, Autonomous Underwater Vehicles (AUVs) will begin to perform missions which have previously been accomplished by attack submarines. AUVs could "...change the nature of undersea warfare..." and by the year 2025 they "...will be as important as manned systems and by 2050 will be the predominant [undersea] warfighting systems." [*HOLZER 90*]

AUVs will perform such missions as underwater surveillance, tracking, minelaying, and possibly offensive antisubmarine warfare missions [*BARKER 86*]. Richard Rumpf, the assistant secretary for research, engineering and systems, in testimony to the House Armed Services Committee in April of 1990, emphasized the U.S. Navy's commitment to AUVs "...for a variety of naval missions, including antisubmarine warfare." [*HOLZER 90*]

Other prominent individuals see AUVs as "mission enhancers" for manned submarines, and feel that they will be "...integrated combat systems which are a part of a submarines sensor suite...." [*BAKER 89*]

Current U.S. Navy schedules project the deployment of AUVs aboard the SSN-21 Seawolf by 2000, and on Improved SSN-688 Los Angeles-class submarines by 2005 [*ROBINSON 86, HOLZER 90*].

Untethered, unmanned submersibles offer significant advantages over conventicnal manned submersible vessels:

- They can perform tasks which are considered too dangerous for humans to accomplish.
- They can operate at tremendous depths, and can maneuver without regard for human physiological limitations.

Along with these advantages come disadvantages, primarily the lack of human control and the inability to intervene in the event of unforeseen problems. To compensate for these shortfalls, a successful AUV must be able to incorporate the advantages of artificial intelligence, real-time control, environmental sensing and maneuverability into a compact, integrated package.

The development of autonomous underwater vehicles which have these characteristics, and are capable of operating under a wide variety of situations and operating conditions, is an area of intense and diverse investigation. A prime problem in fielding an operational AUV is the requirement for "...robust software to maintain vehicle integrity while accomplishing mission goals...." [*BELLINGHAM 89*] At the Naval Postgraduate School (NPS), two AUVs have been constructed in an attempt to address some of this issues.

The first NPS submersible, designated AUV 1, was 27 inches long and displaced less than 20 pounds, which allowed operation in a 40 foot long test tank. This vehicle had an umbilical to provide power to onboard control systems and had no onboard computer system. Vehicle control was provided by an off-hull PC AT microcomputer which transmitted plane commands through radio control equipment to the AUVs dive planes. [*BRUNNER 88*]

In order to allow completely autonomous behavior a second AUV (designated AUV II) is under construction. AUV II will support analysis of vehicle dynamics and control, real-time control systems theory, higher level artificial intelligent processing, robotics and computer architecture for future large AUVs. This research is being conducted as part of a long-term study funded and sponsored by the Naval Surface Weapons Center, White Oak, Maryland [*HEALY 89b*].

A. OBJECTIVES

This thesis covers several tasks in support of the initial launch of AUV II including:

(1) Selection of the Vehicle Control computer and data translation hardware and real-time computer operating systems which meet the requirements specified above

(2) Design, development and testing of the vehicle Guidance subsystem

(3) Development and testing of real-time control software for the control surfaces and main motors on AUV II.

B. BACKGROUND

1. Vehicle Characteristics

The basic layout of AUV-2 is illustrated in Figure 1. The main vehicle body is constructed as a rectangular aluminum box with a beam width of 16 inches, a height of 10 inches, and a length overall (LOA) of 93 inches. It is anticipated that the finished vehicle will displace approximately 370 pounds.

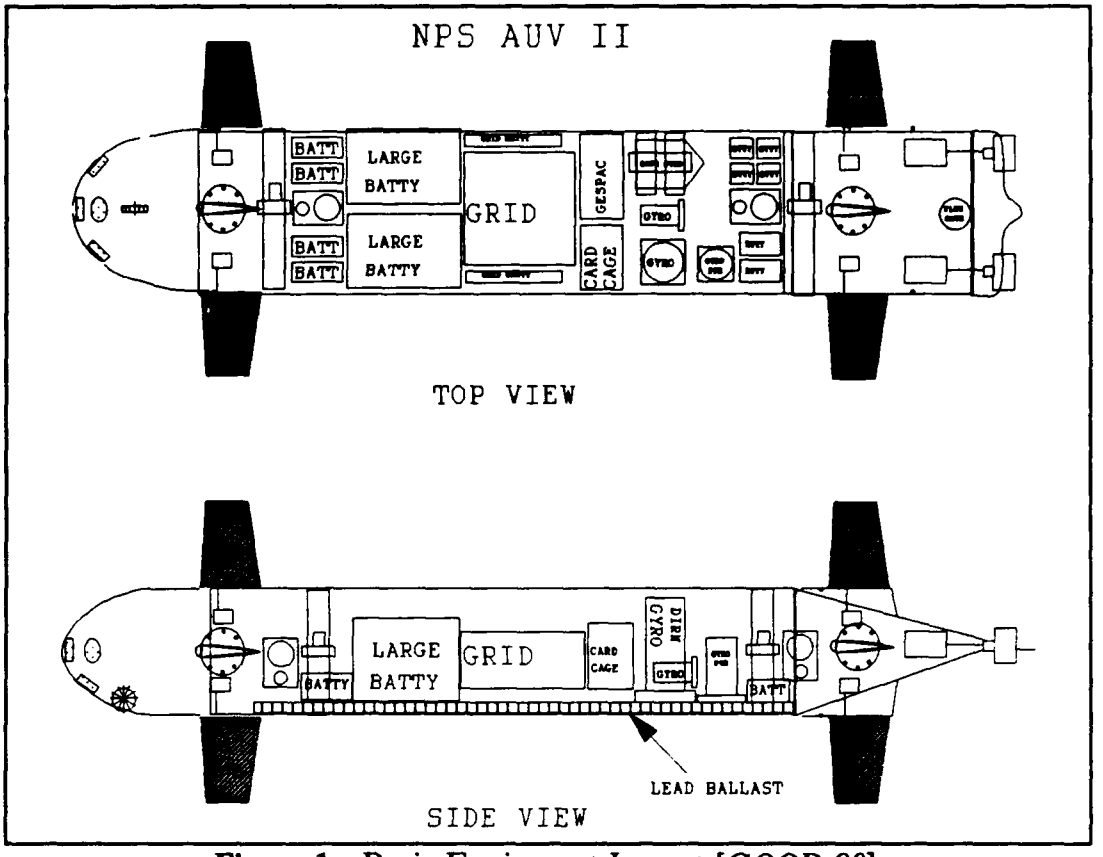

Figure 1 - Basic Equipment Layout [*GOOD 90*]

AUV-2 has independently controllable port and starboard bow planes, and port and starboard stern planes as well as independent fore and aft rudders.

Propulsion is provided by four tunnel thrusters (fore and aft athwartships, fore and aft vertical) and two main screws aft. The tunnel thrusters, combined with the two main screws give the vehicle active control of five degrees of freedom (pitch, yaw, heave, sway and surge) at very low speeds (Hovering Mode). When the vehicle is operating at higher speeds (Transit Mode), all six degrees of freedom (pitch, roll, yaw, heave, sway, and surge) are actively controlled using the main screws for

propulsion and the hydrodynamic forces on the control surfaces to maintain desired pitch, roll and yaw rates. [*KWAK 90*]

There is a sonar system consisting of four individual pencil-beam sonar transducers mounted in a flooded nose, an inertial sensor suite (consisting of a flux gate compass, three rate gyros, three accelerometers, a vertical gyro and a directional gyro), a differential pressure sensing depth cell, a paddle wheel speed sensor, and individual motor RPM sensors. The overall combination of sensors and effectors results in a minimum requirement of 12 separate analog control output signals and 23 different analog input signals for AUV-II (as summarized in Table I).

There were numerous goals established for the development of AUV II. The vessel computer system had to be low cost, utilizing off-the shelf hardware where possible, be readily reconfigurable for different missions, and be expandable to multi-processor capability at a later date. The operating system and control software had to be easy to implement (preferably in the C language), had to use a commercially available operating system, must be able to simulate complex behavior, should support multi-tasking and should be easy to reconfigure for different missions.

2. AUV Software Classification

Software for NPS AUV-II can be divided into two broad classes: Mission Planning software (off-vehicle non-real-time software) and Vehicle Control software (real-time on-vehicle software).

Mission Planning is the process of determining a reference path to be followed by the AUV based on knowledge of pre-existent obstacles, threats and other

Table I - Analog Input and Output Signals of AUV II

ANALOG INPUT SIGNALS	ANALOG OUTPUT SIGNALS
Port and Starboard Bow planes (2)	Control Surface Position (5)
Port and Starboard Stern planes (2)	Propulsor RPM (6)
Fore and aft Rudders (2)	Pitch/Pitch Rate (2)
Port and Starboard main engines (2)	Roll/Roll Rate (2)
Fore and aft horizontal thrusters (2)	Yaw/Yaw Rate (2)
Fore and aft vertical thrusters (2)	Depth (1)
	Sonar Channels (4)
	Speed (1)
	Health and Well-being (4)

possible hazards to the vehicle. The Mission Planner constructs a path as a series of waypoints for the vessel to reach. Development of the Mission Planner for AUV II is described in [*ONG 90*]. This obstacle-free path is downloaded to the vehicle via a detachable RS-232 serial link.

The Vehicle Control software utilizes the path information from the Mission Planner and controls the vehicle from point of origin to mission goal. As the mission progresses, the Vehicle Control system analyzes all input signals from onboard sensors and provides output signals to control surfaces, thrusters and main motors while piloting the vessel to the desired goal.

The Control System Software Architecture for AUV II is illustrated in Figure 2 and the proposed hardware architecture is shown in Figure 3.

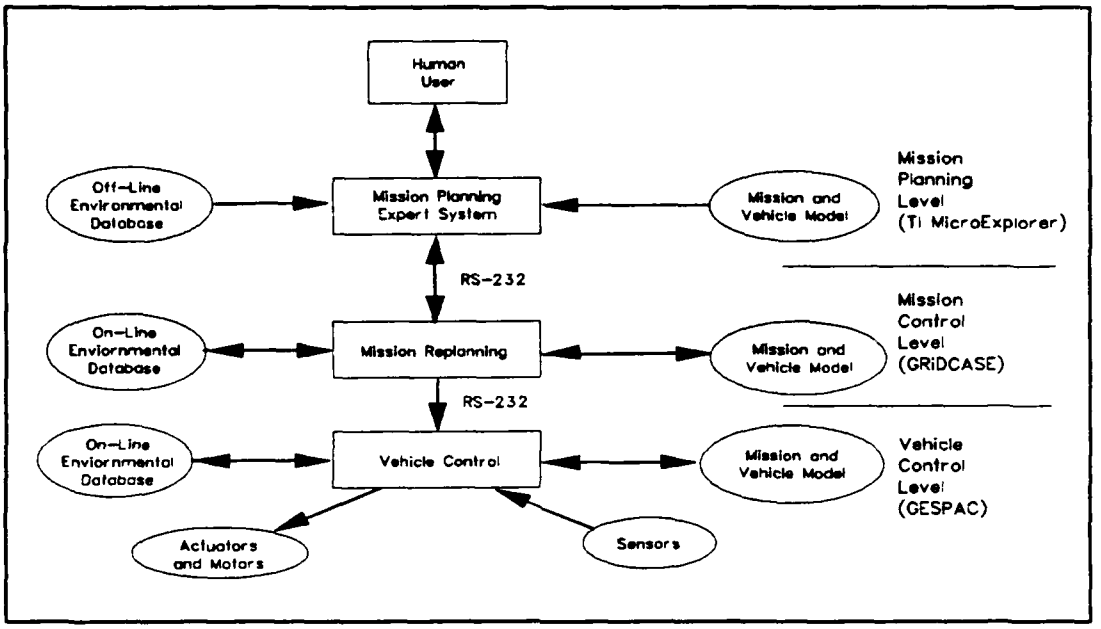

Figure 2 - Control Systems Software Architecture

An off-hull LISP machine (currently a Texas Instruments Micro-Explorer) generates and downloads the Mission Control software to AUV-2 just prior to deployment. The current Mission Control software consists of a series of waypoints to be achieved during the mission. A waypoint is a quadruple made up of the desired three-dimensional position (x,y,z) combined with desired vehicle velocity (v). The quantity of information available in a waypoint is an important consideration in the development of the Local Path-Planner (LPP) and is discussed fully in Chapter 3.

3. Real Time Vehicle Control Requirements

AUV II requires a complex real-time control system which must be able to simultaneously handle a myriad of tasks. These tasks range from the lowest level,

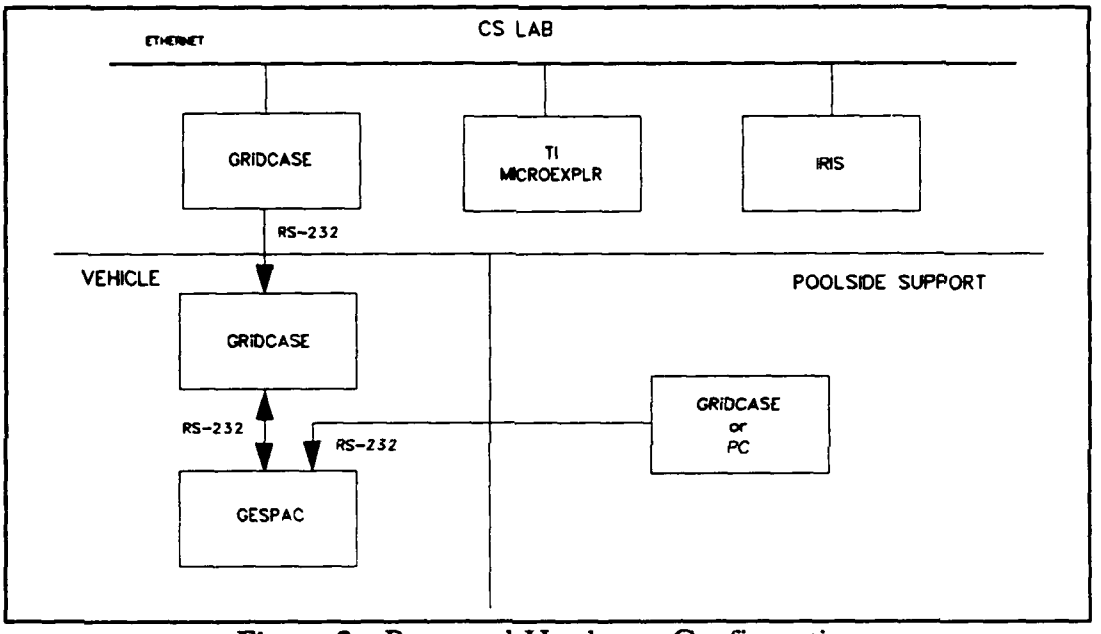

Figure 3 - Proposed Hardware Configuration

such as managing interrupts and processing the sensor input/output specified above, to very high level tasks, such as obstacle avoidance and path-replanning. The development of this type of system is dependent on numerous constraints which are generated by the nature of the problem such as:

- program and mission complexity
- data processing and storage requirements
- control loop timing

as well as constraints generated by the choice of hardware:

- processor speed
- data translation device speed
- communication bandwidth
- compiler and debugger capability

Tests with NPS AUV-I demonstrated that an IBM AT-clone could provide autonomous control for diving plane operation while controlling a single propulsor. With AUV-II, both the number of controls required and their complexity will be increased.

For AUV II the Autopilot control loop must be closed at a minimum of 10 times per second (every 100 msec). The Vehicle Control computer system must be able to multi-task, and should be expandable to allow additional of data translation, memory and processor capability [*HEALY 90*].

4. Guidance Subsystem Requirements

The Guidance subsystem receives waypoints (x,y,z,v) from the Mission Planner. These waypoints correspond to points on a three-dimensional grid being traversed by the vehicle. These waypoints must be fitted to a smooth path, and the path information must be fed to the Autopilot subsystem as series of intermediate positions (x,y,z,v) with heading and pitch angle (θ,ϕ). These intermediate positions must be fed at a rate which will allow smooth operation of the AUV.

There have been numerous methods of trajectory planning developed in the past decade [*BRADY 82, KHATIB 85, SHIN 86*]. These methods were all based on a set path in cartesian space which could be transformed to a set of points which describe the motion through inverse kinematic equations. It has been shown that jerk, the third derivative of position of the desired trajectory, adversely affects control algorithms and therefore should be minimized [*KYRIAKOPOLUS 88*]. It is also

intuitive that in an underwater vessel, jerk can not be tolerated because of the hydrodynamic constraints in place on the vehicle.

Kanayama and Hartman originally proposed the utilization of the derivative of path curvature of cubic spirals[1] as a cost function for path smoothness in a local path planner for an autonomous vehicle [*KANAYAMA 88a*]. The cost function utilized in the preparation of the path between postures is the integral of the square of the derivative of curvature of the path which minimizes jerk.

Kanayama's method of guidance was first implemented on Yamabico-11, an autonomous land-based mobile robot [*KANAYAMA 88b*]. The elegance and simplicity of Kanayama's method was attractive, and appeared to be readily adaptable to AUV control as well.

Kanayama also developed a set of equations providing stable feed-forward tracking control for Yamabico 11 [*KANAYAMA 90*]. His method utilized postures (x,y,θ) as input from the 'Mission Planning' level, and output postures and velocities to the Autopilot. In this thesis, both Kanayama's guidance and tracking control methods have been modified for implementation and use on NPS AUV II.

C. THESIS ORGANIZATION

Chapter II presents a survey of previous work on AUV systems and associated technology. Current operational AUV and Unmanned Underwater Vehicle (UUV)

---

[1] A cubic spiral is a curve whose tangent direction is described by a cubic function of length.

systems are described, and their effect on the hardware and software decisions made for AUV II is presented.

Chapter III provides a detailed description of the vehicle computer hardware and associated low level software. This includes a review of the rationale for the choices made as well as a discussion of the development of the control firmware (data conversion and control) routines which were created for AUV II.

Chapter IV discussed the development and implementation of the Guidance Subsystem including the on-line and off-line local path planners, and the tracking controller. Background information necessary to understand the Guidance problem for AUV II, and the relationship between Guidance and applicable software modules, is also provided.

Chapter V provides a summarization of the overall work, and gives recommendations for possible extension, expansion and improvement. This chapter also discusses a review of the contributions made to autonomous vehicle development.

II. RELATED WORK

This chapter provides an overview of current work in the field of autonomous vehicle control as well as related topics pertinent to the design of the AUV II Vehicle Control Computer and the Guidance subsystem are also presented. The effect of these other projects on the hardware and software decisions made for AUV II is summarized.

A. EDUCATIONAL INSTITUTIONS

1. Texas A&M

Texas A&M University has proposed the use of a modified blackboard system based on "situations" as the structural design paradigm for the knowledge based control architecture in AUVs. These situations are defined as "...the rule sets, domain and declarative knowledge required to make the decisions, judgements and actions required of the reasoning component in the corresponding part of the problem space...." [*MAYER 87*] A prototype implementation of this strategy has been in incorporated on four Symbolics 3640 machines, but the system does not appear to have been implemented on an operational vessel.

2. University of Florida

The University of Florida has done work on a hierarchical control system for autonomous vehicles based on work by Sardis. The core of this system is a three-level "Intelligent Module" controller consisting of a Planner, Navigator and Pilot

which all interface with a Cartographer, and a separate Low-Level Controller [*ISIK 84*]. This system has been modeled, but reference to an operational vessel has not been found.

3. Georgia Institute of Technology

Arkin at Georgia Institute of Technology proposes the use of "Motor Schema" as the basic unit of behavior for navigation [*ARKIN 87*]. These "schema" are multiple concurrent processes which operate in parallel with sensory schema to provide vehicle motion. This differs from Brooks' subsumptive architecture [*BROOKS 88*] in that it avoids layering entirely and instead uses a "souplike" network of schemas which can change dynamically based on the vehicles current needs, perceptions and goals [*ARKIN 87*].

4. University of New Hampshire (UNH)

UNH has been involved in submersible technology since 1977 and has constructed and launched three generations of autonomous vehicles. The current generation of AUVs are the EAVE III vehicles. The EAVE III has a modular, hierarchical computer architecture and utilizes three Motorola 68000 based computers on a VME bus running under the pSOS operating system [*JALBERT 88*]. The lowest level software is coded in 'C' and the upper level is coded in 'LISP'. The vehicle is designed with a blackboard based system for context sensitive higher level mission planning [*CHAPPELL 87*], but these vehicles have also successfully

operated under the NIST NASRAM architecture (MUST vehicles) described in the NIST section below.

5. MIT - Sea Grant College

The Massachusetts Institute of Technology - Sea Grant College (MIT - Sea Grant) has built a small AUV as a software testbed. The vehicle displaces approximately 60 pounds, utilizes three thrusters, has a depth rating of 200 feet and endurance of approximately 2 hours [*BELLINGHAM 90*]. The initial mission selected for the vehicle is to avoid obstacles while proceeding to a stationary target. In pursuit of this goal, the vessel must utilize several "intelligent behaviors" including: homing, obstacle avoidance and information gathering [*BELLINGHAM 89*]. The vehicle utilizes a GESPAC-MPU20 68020 based processor running under the OS-9 operating system. Software is written in C on an IBM AT compatible and cross-complied for the 68020. Prior to the start of a mission, software is downloaded to the AUV and activated through a serial link tether. When the mission is complete, the tether is reattached and data is uploaded to the IBM AT clone for analysis.

The MIT - Sea Grant vessel utilizes a "Layered Control" system adapted from work done by Brooks with land-based robots [*BROOKS 88, 89*]. This architecture is modular in that a mission is broken into discrete behaviors which can output commands. It is reflexive since behaviors respond to immediate sensor readings, thus world models are not required. Behaviors are hierarchically prioritized, and completely asynchronous [*BELLINGHAM 90*]. These modular behaviors are combined to produce intelligent behavior.

6. University of California - Santa Barbara

Kanayama developed a Guidance system for a mobile land robot called Yamabico-11 which utilizes a set of postures (similar to AUV II waypoints) for local path control [*KANAYAMA 89a, 89b*]. This system utilizes portions of cubic spirals as the path to be followed between postures. The cubic spiral has the property of minimizing jerk (the second derivative of acceleration) between postures which is very desirable for an underwater vehicle. This system operated in real-time and is implemented in the C language.

B. INDUSTRY AND GOVERNMENT

1. FMC Corporation

FMC Corporation (FMC) has developed a hierarchical real-time control system architecture where there is an "order of ten" space-time difference between hierarchical levels [*NITAO 85*]. The upper levels in this hierarchy possess a broad, abstract view of the world. Since actions in these higher levels do not affect the narrow, detailed real world of the lower levels, the processing times can be significantly slower than at lower levels.

This method also employs a Reflexive Pilot which does not utilize a memory map of past subgoals. Higher level replanning is only required when the Reflexive Pilot fails to make progress [*NITAO 86*].

FMC is also developing a control system which does path-planning based on 3-D digital maps. They assert that the system can be incorporated into any multi-

level functional architecture. This path planning strategy utilizes a three-tiered system where the User (or Action Planner) furnishes input to a Global Path Planner, which then computes a path to provide to the Local Path Planner for execution [*PARODI 84*].

2. Rockwell

Rockwell is working on an Expert System to "dynamically schedule the allocation of resources" onboard satellites. This system incorporates several "intelligent agents" such as a Satellite Controller, a Planner, and a Subsystem Specialist [*BARRY 88*]. This system is functionally very similar to the FMC system and to the system being developed for NPS AUV II. The Satellite Controller generates an "agenda" (mission) which is provided to the Planner. The Planner generates a plan (path) for the Subsystem Specialist (Pilot) to carry out.

3. Martin-Marietta

Martin-Marietta Aero and Naval System's Mobile Undersea Systems Test (MUST) Laboratory has been working since 1984 to develop a testbed for AUV technology development. They have constructed a 30 foot long vessel which is capable of diving to 2000 feet. The MUST vehicle is controlled by two Motorola 68020 processors on a VME buss utilizing a commercial operating system. All I/O and memory cards are also off-the-shelf circuit boards. Research payloads will drive the control system at a 10 Hz rate. The vessel uses a "...modular, event-driven state

table process system..." designed around the NIST NASREM Real-Time Control System (RCS) [*HEBERT 87*].

4. National Institute of Standards and Technology

The National Institute of Standards and Technology (NIST) proposes the use of the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) a hierarchical control system developed by Albus [*ALBUS 89*]. This architecture incorporates several layers of control, each of which provides commands to the next lower layer. Desired vehicle actions are input at the highest level and decomposed both spatially and temporally [*ALBUS 88*]. As commands are passed to lower levels they are further decomposed (both temporally and spatially) until they become specialized enough for vehicle control.

Each layer has its own sensory processing, world modeling and task decomposition which allow it to sense the environment and take appropriate action. The system supports up to seven or more layers, all of which can share common memory as a blackboard, for map storage, object files, etc.

The command timing structure in the NIST architecture is similar to that used by UNH. Commands are passed at a time interval of approximately 10 times the time interval of the next lower layer [*ALBUS 90*].

The NIST NASREM system architecture was tested on two UNH EAVE III vehicles in a Multiple Autonomous Vehicle (MAUV) experiment funded by the DARPA Naval Technology Office [*ALBUS 88*, *HERMAN 88*]. This experiment

utilized the hierarchical structure of NASREM in conjunction with UNH's lower level control code.

5. Woods Hole Oceanographic Institute

Woods Hole has developed a very capable double tethered vehicle system (ARGO-JASON) [*YOEGER 90*]. This system utilizes 8086 based processors running under the VERTEX operating system. The ARGO-JASON system has the ability to operate in an Auto-Heading or Auto-Depth mode, but does not operate with total autonomy.

6. NOSC

The Naval Ocean Systems Center (NOSC) has configured the EAVE-West vehicle to demonstrate distributable software architecture for AUV plan execution [*DURHAM 87*]. NOSC is also doing work with the possible application of neural network technology to autonomous vehicles [*DURHAM 90*].

7. International Submarine Engineering

International Submarine Engineering Research, Ltd. (ISE) has developed several untethered, remotely controlled submersibles. The first autonomous remotely controlled submersible (ARCS) was designed as a hydrodynamic survey vehicle which follow a series of waypoints. ARCS had a control system consisting of three Intel microprocessors on a common bus utilizing common memory [*THOMAS 85, JACKSON 84*].

DOLPHIN is a diesel powered UUV designed by ISE. It was originally developed as an offshore hydrographic mapping vessel, but has since been revamped for use as a search and survey vehicle, advanced hydrographic survey and seismic exploration. The vehicle control computer on DOLPHIN uses the GESPAC G-96 bus with Motorola 68010 and 680008 processors. ISE conducted an extensive market survey, and chose the GESPAC system based on price, performance, size ruggedness and availability [*WIETZEL 87*].

Because of the increased complexity of the real-time control systems on autonomous submersibles, ISE is pursuing the development of a complex object-oriented real-time control system. This system bases system operation on events and actions and utilizes a simple, efficient preemptive scheduler [*ZHENG 88*]. This scheduler is extremely modular, and can be adapted to a variety of different hardware architectures, and has been tested on Intel based PCs and Motorola 68000/68010 based GESPAC systems.

C. SUMMARY

There are very few operational underwater vehicles which are truly autonomous. A summary of the capabilities of vessels which are truly autonomous is given in Table II. Of the AUVs listed, the MIT Sea Sprite most nearly approximates AUV II in size and intended use.

Table II - Current AUV capability [*after BELLINGHAM 90*]

VEHICLE	Thrust	Comp	Sens	OA
ARCS (ISE)	1f	3(68010)	D	yes
EAVE III (UNH)	2f,2v,2l	3(68000) 1(68020)	D,T	yes
Sea Squirt (MIT)	2f,2l	1(68020)	D,T	dev
MUST (Martin-Marietta)	1f,2v	2(68020)	D	dev?
EAVE-West (NOSC)	2f,1v	1(8080)	D,P,T	no

Thruster Type: f=forward, v=vertical, l=lateral

Computer Type: qty(processor type)

Sensors: D=depth, T=temperature, P=photography

Obst. Avoid: yes, no, dev=under development

The following observations were based on known AUV systems, and were central to the choices made for the AUV II Vehicle Control Computer system:

- There does not appear to be a clear standard for real-time operating systems for AUVs.
- The 'C' language appears to be the preferred language for control-level coding.
- Several AUVs utilize the GESPAC Motorola-based architecture.
- Most vehicles have a modular, hierarchical control structure, with each level operating about ten times faster than the layer which lies lower in the heirarchy.
- The cubic spiral system Kanayama created for Yamabico-11 is readily adaptable for use in AUV II.

III. EVOLUTION OF ONBOARD COMPUTER SYSTEM

This chapter presents the evolution of the Vehicle Control Computer system for AUV II. The Vehicle Control Computer system consists of a GRiDCASE 1535 EXP based on a 12.5 MHz Intel 80386 processor and a GESPAC system centered around a 25 MHz Motorola 68030 processor.

A. GRiDCASE 1535 EXP

A survey of available systems which could meet the increased computational requirements for AUV-II conducted prior to the start of this thesis resulted in the purchase of the GRiDCASE 1535 EXP (GRiDCASE). This is a 12.5 Mhz 80386-based laptop computer which runs under a version of MS-DOS modified by the GRiD corporation. The purchased configuration included an 80387 math coprocessor, 4 Mbyte of RAM, a 40 MByte hard drive, a 2400 baud internal modem, and an expansion tray which allowed for the addition of one AT-compatible expansion card and one XT compatible expansion card [*GRID 88*].

1. Data Translation Boards

Two Data Translation (DT) cards were purchased to provide analog-to-digital and digital-to-analog capability for the GRiDCASE. The first was a DT2821 High-Speed DMA combined analog-to-digital/digital-to-analog converter and the second was a DT2815 digital-to-analog converter [*DATA 89*]. The DT2821 provided 16 channels of digital input/output, 2 channel of 12-bit digital-to-analog outputs and

16 single-ended (or 8 differential) input analog-to-digital inputs. The DT2815 provided 8 channels of 12-bit digital-to-analog output. The combined total capability of the system (which used both available card slots on the GRiDCASE) was:

- 10 channels digital-to-analog output
- 8 channels analog-to-digital input
- 16 channels digital input

As can be seen from Table I this combination did not meet the initial specified data translation requirements, but it was thought that satisfactory operation could be achieved if some input signals were multiplexed and if control surfaces (such as bow planes and stern planes) were ganged together. (In fact, multiplexing would be undesirable since demultiplexing of the input signals would require additional CPU time during execution of the primary control loop.)

Test programs were written to verify proper operation of the DT boards. Simple input/output tests were successful, but attempts to utilize the DMA capability of the DT2821 failed repeatedly. Several versions of code were written in an attempt to get proper DMA operation. Numerous calls to technical support personnel at both Data Translation and GRiD over a period of months were fruitless. Each vendor attempted to point to the other as being the source of the problem. The final result was that DMA did not function properly.

2. Real-Time Operating System

Because of the timing requirements for the Vehicle control computer (control loop must be executed at a minimum 10 Hz rate) it was decided that the

new system should be able to multi-task (at least to the point of allowing the Autopilot to operate independently of all other modules). This posed an interesting problem, in that the GRiDCASE computer was operating under MS-DOS, which does not readily support multi-tasking.

a. TSR Program

A simple evaluation program was written to determine if a single control program could operate at a specific time interval (see Appendix A). The program written was a terminate-and-stay-resident (TSR) program that operated at a 100 msec frequency, while allowing another program to operate in the foreground at all other times [*DETTMAN 88*]. This program took advantage of DOS interrupt 28h (the timer interrupt) and showed that two programs could be operated successfully under MS-DOS.

The primary disadvantage to the TSR method was that all procedures and code which were included in the TSR loop had to be completely devoid of DOS level calls (because the TSR interrupt INT 28h operates below the DOS level). If this method were chosen for Full Scale Development, all C function calls would have to be evaluated to ensure proper operation and many would have to be rewritten. As an example, the C printf would not work because it utilizes DOS calls when printing to screen, and so would have to be modified to use BIOS level calls for screen input/output.

b. Regulus Operating System

In order to provide a true multi-tasking capability, it was decided to purchase and evaluate the Regulus operating system. Regulus is a UNIX-like operating system that provides such features as real-time tasks, shared memory data segments and user access to physical memory [*ALCYON 86*]. Regulus was purchased and installed on a GRiDCASE EXP-1535. The installation procedures for Regulus assumed that the host machine was an 80386 machine operating under true MS-DOS. Because the GRiDCASE operates under a modified version of MS-DOS there were some minor problems with installation. The Regulus operating system is relatively new, and the user manuals offer little insight into the systems multi-tasking capabilities. Regulus' customer support system was able to offer little help on applications, and gave the impression that system performance would be significantly degraded in a multi-tasking mode using high-speed DMA.

3. Hardware Problems

There were now several distinct problems with the original hardware and software purchased for AUV-II:

- There was no room for expansion (both card slots were filled).
- There were inadequate A2D and D2A channels for required input and output signals.
- A maximum of two tasks could operate under DOS "simultaneously" (one a TSR), and the TSR task must be entirely devoid of DOS system level calls.
- The chosen machine was relatively slow (12.5 Mhz).

- Since the converter boards were from a different manufacturer (DT) than the computer (GRiD), compatibility problems were difficult to resolve.
- The multi-tasking operating system (Regulus) might eventually work, but once again there were incompatibilities due to the modified MS-DOS operating system used by GRiD.

Because of these problems, a new survey for a real-time control computer system and operating system was conducted. A true multi-tasking operating system was desired. In addition, the new system must be expandable to multi-processor capability and have the ability to add additional data conversion capability. A final requirement was that other autonomous vehicle projects be using the system, to ensure that the system would work in an undersea environment, and to allow transfer of information and low-level source code.

B. GESPAC

After the survey of literature discussed in Chapter 2 was complete, several organizations were contacted directly about appropriate hardware. Discussions with the Monterey Bay Aquarium Research Institute (MBARI), International Submarine Explorations (ISE), Naval Ocean System Center (NOSC), the University of New Hampshire (UNH) and the Sea Grant College at Massachusetts Institute of Technology (MIT - Sea Grant) resulted in the choice of GESPAC.

GESPAC offers a variety of Motorola 68000 based systems. The GESPAC systems operate under the OS-9 commercial operating system produced by Microware [*GESPAC 88*]. OS-9 is a true multi-tasking operating system [*MICROWARE 87*] and is currently in use in the MIT - Sea Sprite AUV, (allowing

possible use of MIT Sea Sprite data conversion code). GESPAC manufactures their own data conversion cards which minimizes the possibility problem of hardware incompatibility [*GESPAC 89*]. Both ISE and MIT have successfully utilized GESPAC systems on autonomous vehicles. It was decided to purchase two GESPAC systems with similar data translation capabilities for AUV II.

1. Target System

The system to be placed on AUV II is designated the target system . It utilizes a GESPAC MPU30HF main processor board with a Motorola 68030 CPU and 68882 FPU operating at 25 MHz. This board is configured with 2.5 Mb of on-board RAM and has room for 4 Mb of EPROMS. To meet the data translation requirements specified in Chapter 1, four additional boards were required: an ADA1 (16 SE/8 DE analog-to-digital input channels, 4 digital-to-analog output channels); an ADC2B (16 SE/8 DE analog-to-digital input); a DAC2B (8 digital-to-analog output channels); and a MFI card (2 serial ports, 2 8-bit parallel ports). A GESPAC SCSI hard disk controller card and a 200 MB Conner peripheral hard drive will be utilized for data storage. The six GESPAC cards are placed in a 12 card rack to be mounted in AUV II, and not only met all known data processing requirements, but also allowed six additional slots for future expansion.

For the target system a PC Bridge cross-compiler and source level debugger was purchased. PC Bridge allows an IBM compatible machine to serve as a terminal and data storage device for the development system, which currently has no disk storage. With this system, control software for the vehicle can be written in

C on an IBM PC compatible machine, compiled and debugged using PC Bridge, and then downloaded via an RS-232 serial link to the vehicle for testing. When the vehicle is ready for launch, mission software can be downloaded to the vessel using PC Bridge and a serial link.

2. Development System

The off-hull OS-9 system has been designated the development system . It utilizes identical processor and data translation boards as the target system. This allows for any of the boards in this system to be transferred to the target system in the event of a board failure.

In addition, this system is configured with a 40 MByte hard-drive and the OS-9 development compiler and debugger. This system operated entirely under OS-9 (PC Bridge operates under MS-DOS) and therefore cross-compilation is not required.

A second advantage of the development system is that when the low level control code has been adequately debugged, this system is ideally suited to prepare Motorola S-records which can be burned into EPROMs and placed on the MPU30HF processor board. This will free up RAM, and speed up the downloading of mission control code.

C. AUV II DISTRIBUTED COMPUTER SYSTEM

The AUV II has a distributed computer control system divided into three major components:

(1) Vehicle Control Computer (VCC) - onboard computer which provides active vehicle control at the lowest level. It passes information to/from the Mission Replanner.

(2) Mission Replanner - onboard computer which performs recalculation of mission plan as required. Provides mission plan to VCC as a series of waypoints. Receives vehicle feedback and sensor data necessary for mission replanning from the VCC.

(3) Mission Planner - off-hull computer which constructs mission based on global world model and data. This is the only computer with direct operator interaction. Downloads mission to Mission Replanner.

The current configuration utilizes the GESPAC as the Vehicle Control Computer and a Texas Instruments MicroExplorer as the Mission Planner. There is no software for the Mission Replanner, so the mission plan is downloaded directly into the Vehicle Control Computer via an RS-232 serial connection.

The final configuration of the on-board computer system will utilize both the GRiDCASE and GESPAC systems. The GESPAC will continue to be the Vehicle Control Computer, performing all actual real-time vehicle control and data translation functions. The GRiDCASE will be used as the Mission Replanner, receiving the Mission Plan as a series of waypoints via the RS-232 serial link from the off-hull Mission Planner and providing them to the Vehicle Control Computer as required. In addition, if obstacles are encountered while traversing from waypoint to waypoint, the Mission Replanner will perform path planning functions similar to

the Mission Planner to create a modified Mission Plan in real-time without operator interface.

D. DATA TRANSLATION SOFTWARE

The GESPAC data translation boards are provided without associated data translation software. In order to properly test the vehicle control software and hardware, it was necessary to write the necessary data conversion routines. The code for these data translation and conversion routines is provided as Appendix B.

1. Digital-to-Analog Routines

Each GESPAC digital-to-analog board is assigned a specific hardware address on the G-96 bus. Individual devices are assigned unique channels on a specific board. As an example, the STBD_MAIN motor is assigned to channel 0 of the DAC2B board.

The user routine accesses the desired control device by a call to a function specifying a individual board type with a channel number and voltage to be written to that channel (e.g. DAC2B(STBD_MAIN,1000)).

2. Analog-to-Digital Routines

The analog-to-digital routines work in a similar manner. The user routine specifies a specific board and channel to be read (e.g. ADA1(STBD_MAIN_RPM)). The analog-to-digital routine reads the specified channel and returns an integer value corresponding to the analog value read.

E. CONTROL OF HYDRODYNAMIC SURFACES

The hydrodynamic control surfaces on AUV II are being driven by Airtronics 94510 model airplane control servos. This type of servo requires a pulse-width modulated control voltage to properly position the device. The servos are powered by an eight-channel pulse-width modulator (PWM) designed by an NPS staff electronics technician. The schematic diagram for this device is provided in Figure 4.

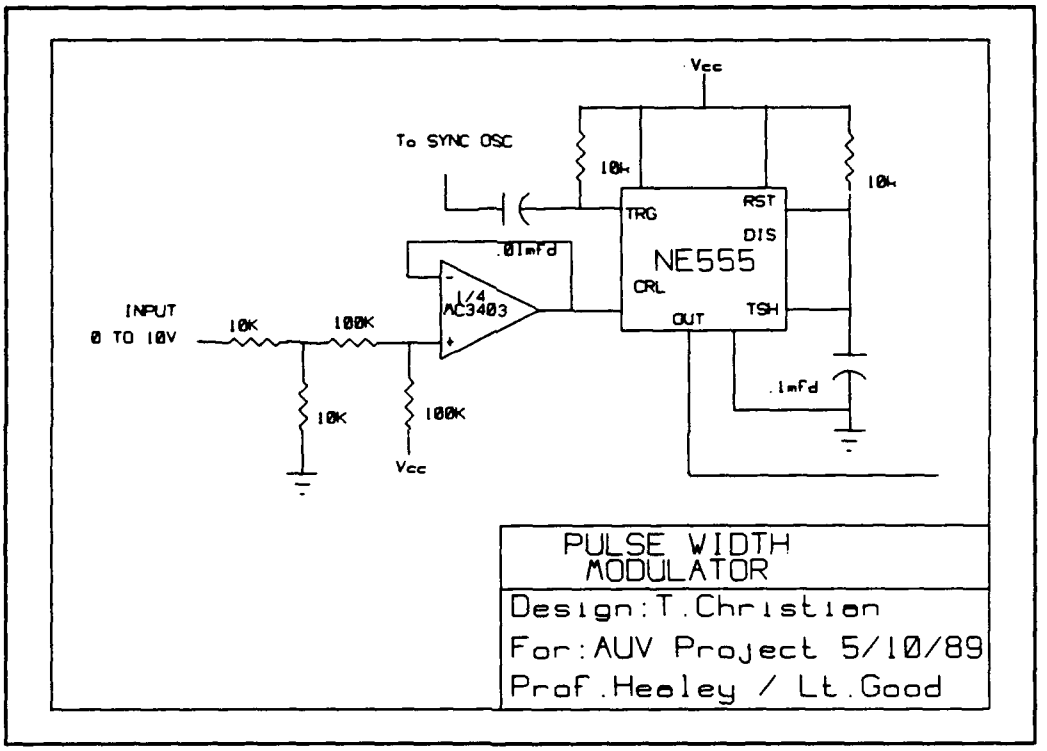

Figure 4 - Pulse-Width Modulator Schematic

The PWM receives a zero to ten volt control signal from the digital-to-analog boards on the GESPAC control computer system and outputs a 0.5 to 1.5 millisecond pulse to the control surface servo motors. The pulse width is a nominal 1 millisecond

with a 5 volt input signal. The output is exponentially non-linear but highly repeatable.

Output pulses are triggered by negative going synchronizing oscillator input signals. This signal releases an internal shorting transistor across the output timing capacitor. The output capacitor charges exponentially through the timing resistor until the output voltage reaches the value of the control input voltage, resulting in an exponential relationship between input voltage and output pulse width. This pulse-width modulated voltage is provided to the appropriate control surface servo motor.

1. Servo Control Testing

Tests were conducted using two different servos and all eight channels of the PWM. The servo test setup is shown in Figure 5. A variable-voltage DC power supply was connected to the PWM control line of each channel. The power supply voltage was varied to achieve control surface positions from -45 to +45 degrees in 15 degree increments. The voltages were checked in both increasing and decreasing directions to verify that the direction of approach did not matter.

The data recorded was extremely stable. The power supply voltage varied less than 20 mV for all combinations of servos and channels. The servo data obtained is provided in Table III and is graphed in Figure 6.

It was expected that the PWM output would be exponentially non-linear because the output pulse-width is determined by the charging of the capacitor connected to TSH of the NE455 (see Figure 4). Experimental results confirmed this hypothesis and the exponentially non-linearity can be seen in Figure 6.

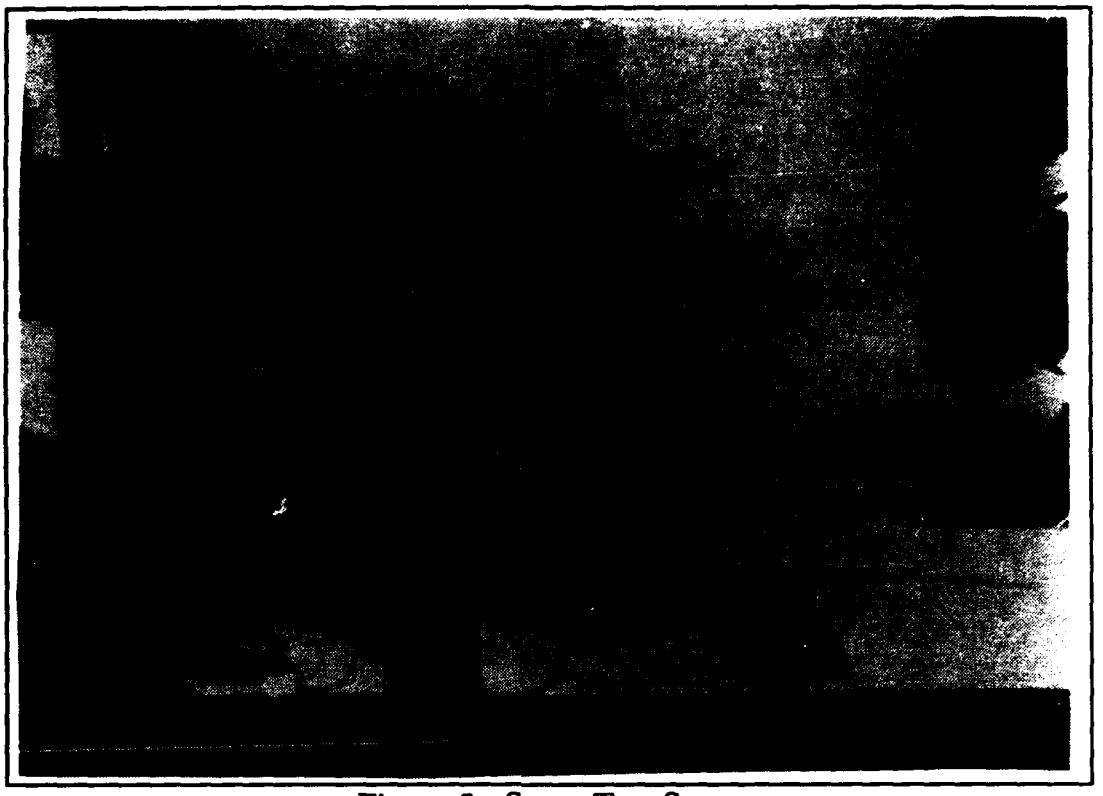

Figure 5 - Servo Test Setup

Table III - Servo Test Data

Angle	Voltage
-45	0.270
-30	1.565
-15	2.835
0	3.855
+15	4.680
+30	5.380
+45	6.020

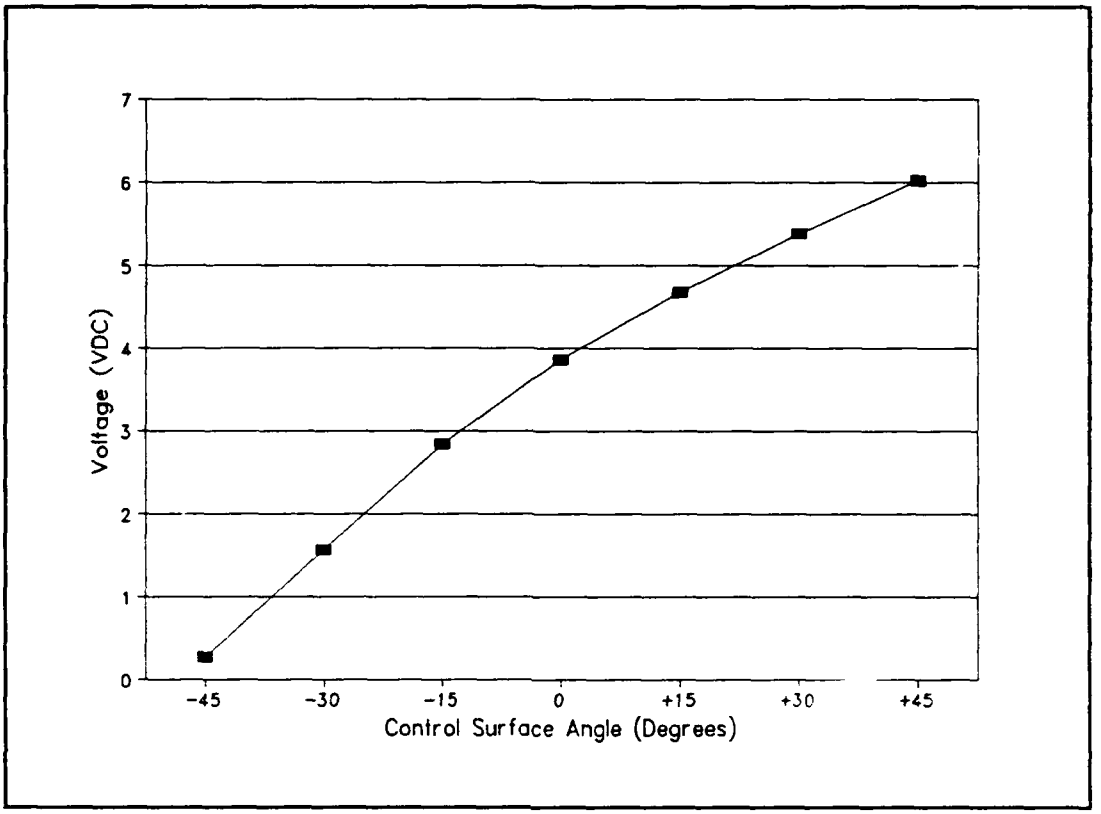

Figure 6 - Raw Servo Control Voltage

2. Servo Control Signal Calculation

The program CALC_SVO.C (see Appendix C) was written to create a look-up table of digital-to-analog signals which provide linear control surface response over the range from -45 to +45 degrees in one degree increments. The program does linear extrapolation of the data contained in Table III and writes to a look-up table file called "servo.dat".

The linear interpolation is based on the equation:

$$V_{des} = \frac{V_{np} - V_{pp}}{Inc} * (Ang_{des} - Ang_{pp}) + V_{pp} \qquad (3\text{-}1)$$

where V_{des} is the desired voltage, V_{np} is the voltage at the next data point, V_{pp} is the voltage at the last data point, Ang_{des} is the desired angle, Ang_{pp} is the angle at the last data point, and *Inc* is the absolute difference between data points (in degrees).

3. Manipulation of Control Surfaces

The Autopilot manipulates the control surfaces through the function **send_servo** which takes two arguments:

- SURFACE - the desired control surface to reposition
- ANGLE - the desired position of that surface

send_servo does a table look-up of the appropriate digital-to-analog signal in "servo.dat" based on the desired position input. This linearized value is passed to the control surface digital-to-analog channel, which then sends the proper control voltage to the PWM. The control signal is pulse-width modulated and sent to the control surface which is positioned to the desired angle.

The calculated table values were compared with actual position and the resultant deviation was less than one degree from desired position over the entire control surface input range (-45 to +45 degrees).

F. MAIN MOTOR CONTROL

The two main propellers on the submersible are driven by Pittman 14202 (WDG #3) 24 Volt DC servo motors [*PITTMAN 87*]. These motors are fed from a motor controller designed and fabricated at NPS by a staff electronics technician. The schematic diagram of the controller is provided as Figure 7.

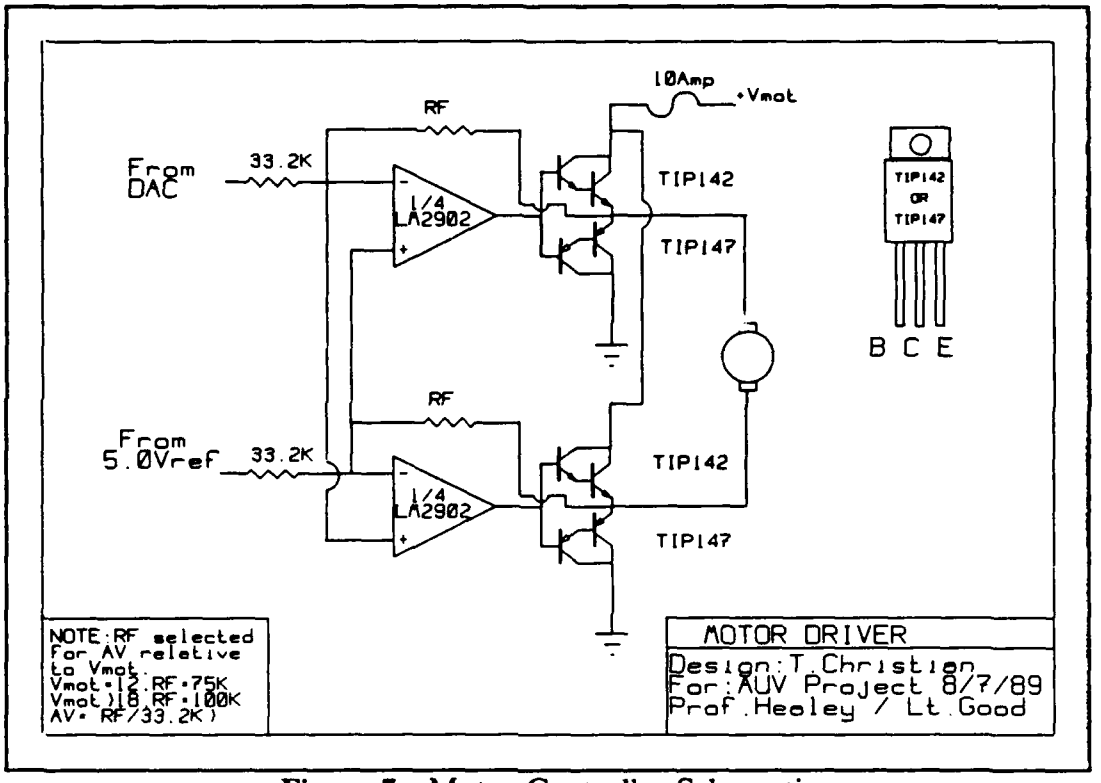

Figure 7 - Motor Controller Schematic

The circuit is a voltage amplifier with a bridge output. Two identical operational amplifiers channels are connected 180 degrees out of phase to Darlington output power transistor pairs which drive the motor.

The motor controller receives zero to ten volt control signals from the GESPAC digital-to-analog converters. A zero volt control signal results in maximum negative voltage output (corresponding to maximum astern RPM) and a ten volt control signal results in maximum positive output being applied to the main motors (corresponding to maximum ahead RPM). A five volt control signal results zero voltage output and the motor remains stationary.

The current Autopilot outputs RPM commands to the main motors (vice outputting velocity commands) [*RILING 90*]. To achieve these RPM commands, a conversion routine was created which converts desired RPM values into appropriate voltage counts for the digital-to-analog boards, which then provide control voltages to the motor controllers to achieve the desired RPM.

1. Main Motor Control Testing

A system test tank was set up to calibrate the main motor controller. The motor test unit was designed to closely approximate the expected vehicle operating conditions and included a motor, propeller and shaft, and Kort nozzle as shown in Figure 8.

A DC power supply was connected to provide motor drive power in the voltage range expected during vehicle operation (20 to 30 VDC) and a variable DC power supply was connected to the control voltage input of the controller. A strobo-tach was used to monitor actual shaft RPM.

The variable DC power supply was adjusted to achieve motor speeds from zero to 700 RPM in both the ahead and astern directions. 700 RPM was chosen as the maximum value because motor current was approaching the five ampere limit of the test equipment circuitry. The test was conducted with motor drive supply voltages of 20 VDC, 24 VDC and 30 VDC.

The control voltage required to achieve a desired motor RPM value was surprisingly independent of motor supply voltage. The control signal required deviated less than 20 mV as the main motor supply was varied from 20 to 30 volts.

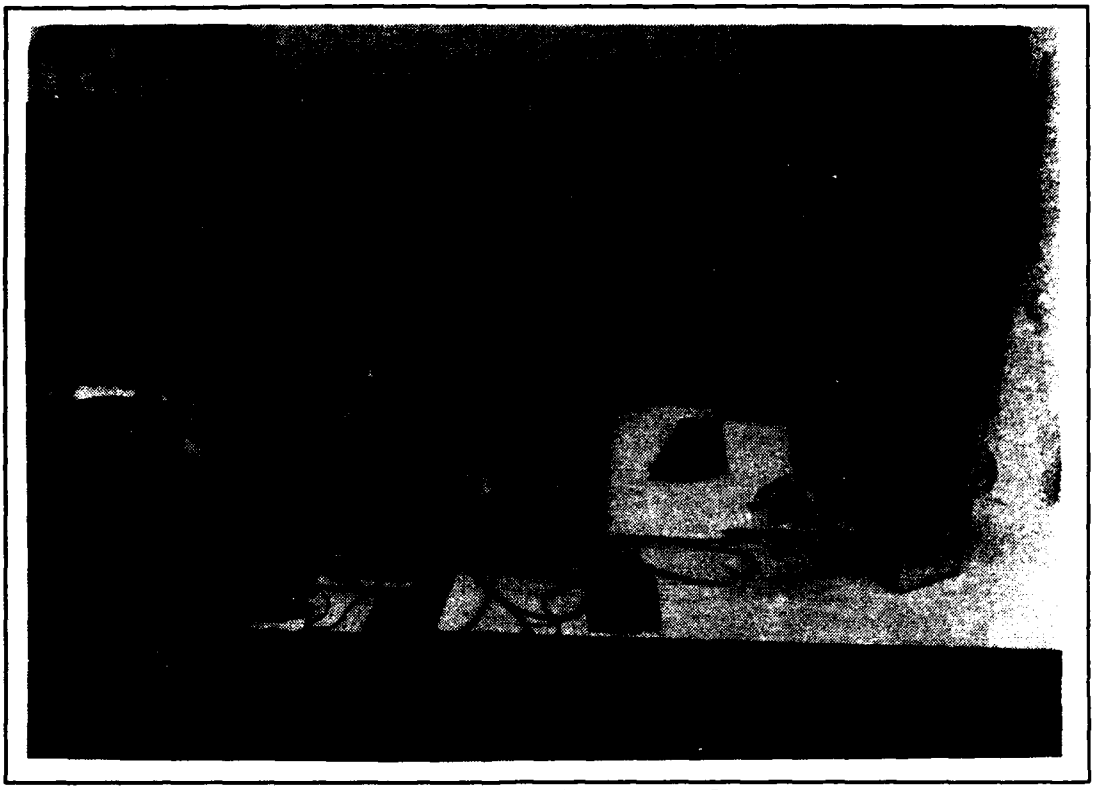

Figure 8 - Main Motor Test Setup

The average values obtained from these tests are recorded in Table IV and are graphically shown in Figure 9.

2. Main Motor Control Signal Calculation

Since the control voltage to RPM relation is non-linear, a software look-up table was needed to linearize the relationship.

CALC_MN.C (Appendix D) creates a lookup table of digital-to-analog signals which provides linearized response in ten RPM increments from zero to 700 RPM in both the ahead and astern directions. This program does linear interpolation of the data contained in Table IV and writes to a look-up table contained in the file "main_mtr.dat".

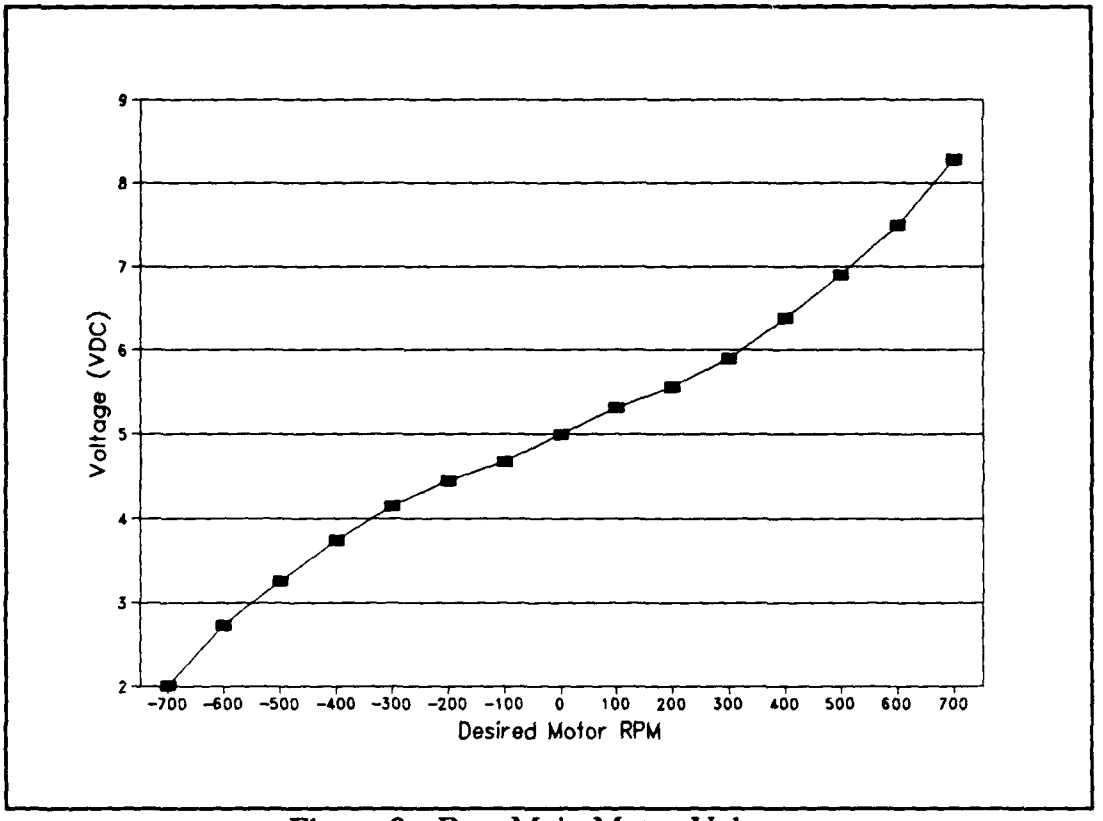

Figure 9 - Raw Main Motor Voltages

Table IV - Main Motor Data

RPM	Voltage (Ahead)	Voltage (Astern)
100	5.319	4.670
200	5.558	4.450
300	5.902	4.140
400	6.387	3.739
500	6.898	3.250
600	7.498	2.730
700	8.280	2.010

The linear interpolation is based on the equation:

$$V_{des} = \frac{V_{np} - V_{pp}}{Inc} * (RPM_{des} - RPM_{pp}) + V_{pp} \qquad (3\text{-}2)$$

where V_{des} is the desired voltage, V_{np} is the voltage at the next data point, V_{pp} is the voltage at the last data point, RPM_{des} is the desired RPM, RPM_{pp} is the RPM at the last data point, and *Inc* is the absolute difference between data points (in RPM).

3. Control of Main Motors

The Autopilot controls main motor speed through a call to **MAIN_RPM** which takes two arguments:

- MOTOR - the desired main motor (Port or Stbd)
- RPM - the desired motor RPM (-700 to +700)

MAIN_RPM normalizes the requested RPM (to 0-1400 RPM) and does a table look-up of the appropriate digital-to-analog signal in "main_mtr.dat" based on the desired motor RPM. This linearized value is passed to the appropriate main motor digital-to-analog channel, which then sends the proper control voltage to the motor controller. The control signal is amplified and sent to the main motor.

The calculated table values for the main motors were checked against actual motor RPM at supply voltages from 20 to 30 VDC over the main motor input range (700 RPM astern to 700 RPM ahead) with an observed deviation of less than one percent.

IV. GUIDANCE SUBSYSTEM

The purpose of the Guidance subsystem is to provide desired waypoint, heading and velocity input to the Autopilot subsystem in a manner which will minimize jerk. In order to achieve this goal, Guidance requires reference waypoint input from the Mission Planner and current vessel parameters from the Navigation sub-system.

'Reference' parameters are those received directly from the Mission Planner. 'Current' paramters are values which are calculated by Navigation during the most recent iteration. 'Desired' parameters are those values fed to the Autopilot after processing by Guidance. More complete definitions of these terms can be found in Appendix E.

A. INPUTS AND OUTPUTS

The Guidance Subsystem provides local path-planning and tracking control for AUV II. It receives as input a file of waypoints consisting of three dimensional grid position and velocity (x,y,z,v) from the Mission Planner. These waypoints are used to construct a smooth path between waypoints. This smooth path is broken into a series of discrete postures (x,y,z,θ,ϕ,v) which are then output to the Autopilot, one posture at a time.

B. SUBSYSTEM COMPOSITION

The Guidance Subsystem for AUV II is comprised of three major functional parts:

(1) **OFF-LINE LOCAL PATH PLANNER** - an off-line program (see Appendix F for source code) which extracts and stores desired curve information for all allowable combinations of waypoints.

(2) **ON-LINE LOCAL PATH PLANNER** - on-line program (see Appendix G for source code) that steps through a given cubic spiral, providing reference postures to the tracking controller at a user specified frequency.

(3) **TRACKING CONTROLLER** - on-line program (also in Appendix G) which provides navigational corrections to the reference postures from the on-line local path planner to create desired postures. The desired postures are fed to the Autopilot.

Each component of the Guidance subsystem is described in detail in a subsequent portion of this chapter.

The Guidance system algorithms presented herein perform local-path planning and Tracking Control for two-dimensions. Application to the third dimension (depth) is discussed at the end of this chapter.

C. INTERFACE WITH OTHER SOFTWARE MODULES

It is useful to divide the computer systems onboard an autonomous vehicle into individual modules which perform specific functions. This approach has been followed in a number of AUVs and Remotely Operated Vehicles (ROVs) [*CROWLEY 85, ALBUS 90, RUSSELL 86, BELLINGHAM 90, ZHENG 90*]. For AUV II, the onboard software is divided into the following Modules:

- Mission Replanning
- Navigation
- Collision Avoidance
- Obstacle Recognition
- Local Path-Planning (LPP)
- Autopilot
- Data Collection
- Hovering

The inter-relation of these modules is illustrated in Figure 10. Three of these modules/subsystems interact directly with the Guidance subsystem:

- Mission Planner
- Navigation
- Autopilot

The function of these modules and their interaction with the Guidance subsystem are described below.

1. Mission Planner

The Mission Planner is responsible for finding a safe path between the starting position and the goal position utilizing the information available from the off-line world model. The path must minimize distance while at the same time taking into account other factors such as mission duration, threat analysis, and other possible hazards. The output of the mission planner must be readily usable by the onboard computer systems.

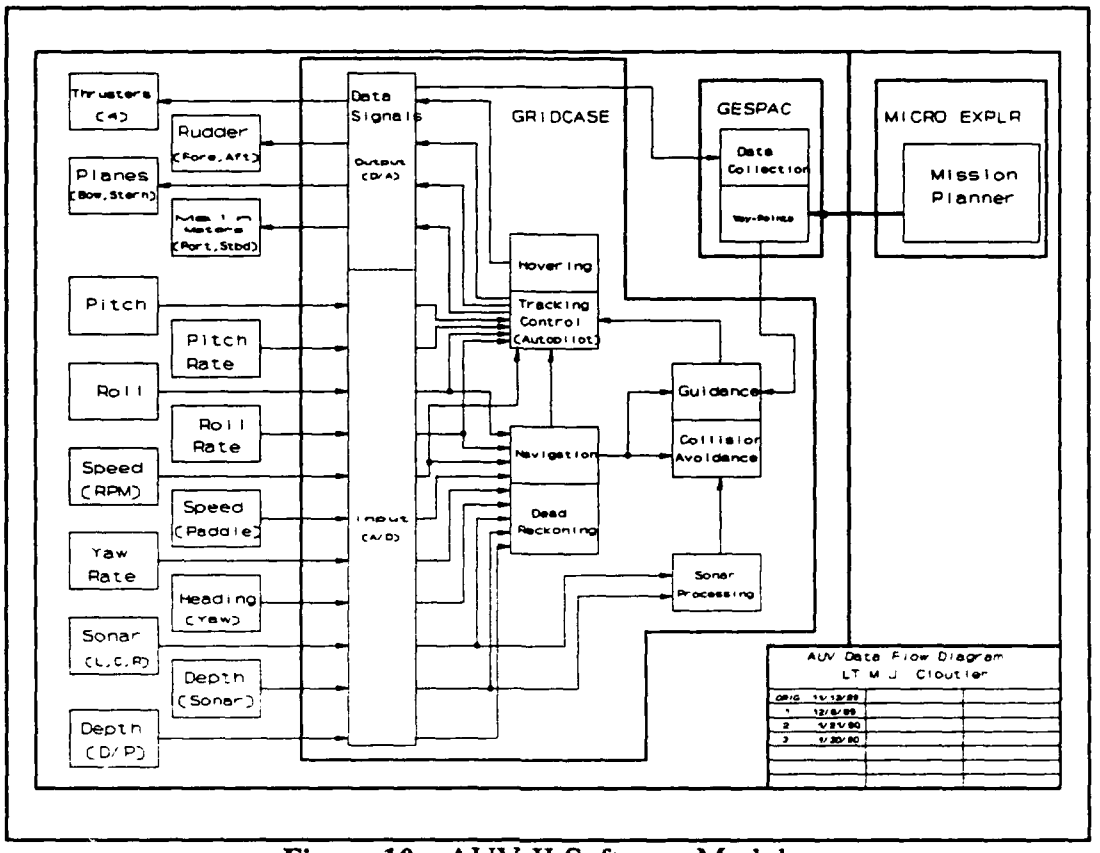

Figure 10 - AUV II Software Modules

Because the paths that an autonomous vehicle must follow are complex and subject to change, real-world data must be obtained and utilized to arrive at satisfactory solutions [*MOZIER 87*]. AUV II will contain vehicle models and mission models at various levels, and will have on-line environmental databases to ensure proper data input for all of the above modules [*HEALY 90*].

The choice of an appropriate global or world model is essential to designing and implementing a successful navigation system and as a consequence most mobile robot planners are strongly influenced by the world modeling method employed. In addition, selection of appropriate information to be passed to the

vessel can greatly simplify the design of the Mission Planner. "The key to a successful modelling technique lies in defining a conceptual model that adequately describes the operating environment *for the tasks being performed* and <which> lends itself to computer implementation...." [*MOZIER 87*] By utilizing a grid structure for the world model, formal search methods such as A* and depth-first can be employed in conjunction with appropriate heuristics. The current Mission Planner for AUV II is based on a three-dimensional rectangular grid and uses several different heuristic search methods [*ONG 90*].

a. Grid Size Determination

The grid has identical x and y dimensions, with a smaller z dimension. A two-dimensional representation of this grid is illustrated in Figure 11. For the purposes of software testing he grid is sized based on estimated vessel maneuverability. The actual values must be validated during initial trials.

The Mission Planner assumes that the AUV is traveling on a three-dimensional grid. The algorithm utilized by the Mission Planner in calculating a desired path assumes that the AUV can only traverse directly from one waypoint to another waypoint on the grid (it cannot stop at intermediate points off of the grid), and that when it arrives at the next waypoint, it will be heading in a cardinal direction (North, South, East or West). This type of maneuver is illustrated in Figure 12 . The position and direction assumptions are the critical link between the algorithms used by the Guidance Subsystem and the path prepared by the Mission Planner.

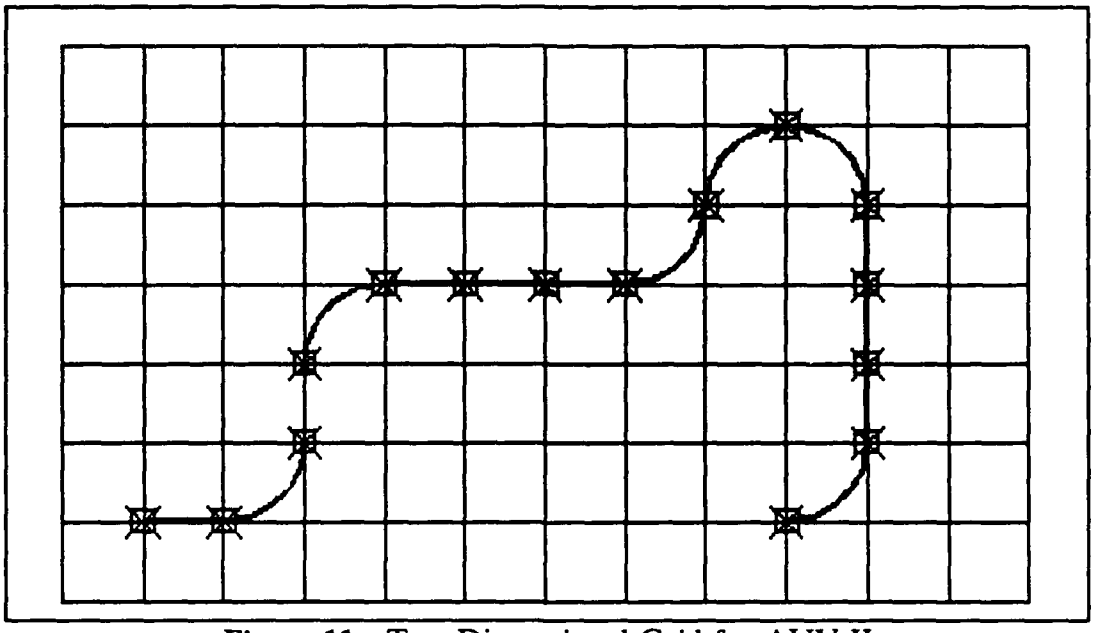

Figure 11 - Two-Dimensional Grid for AUV II

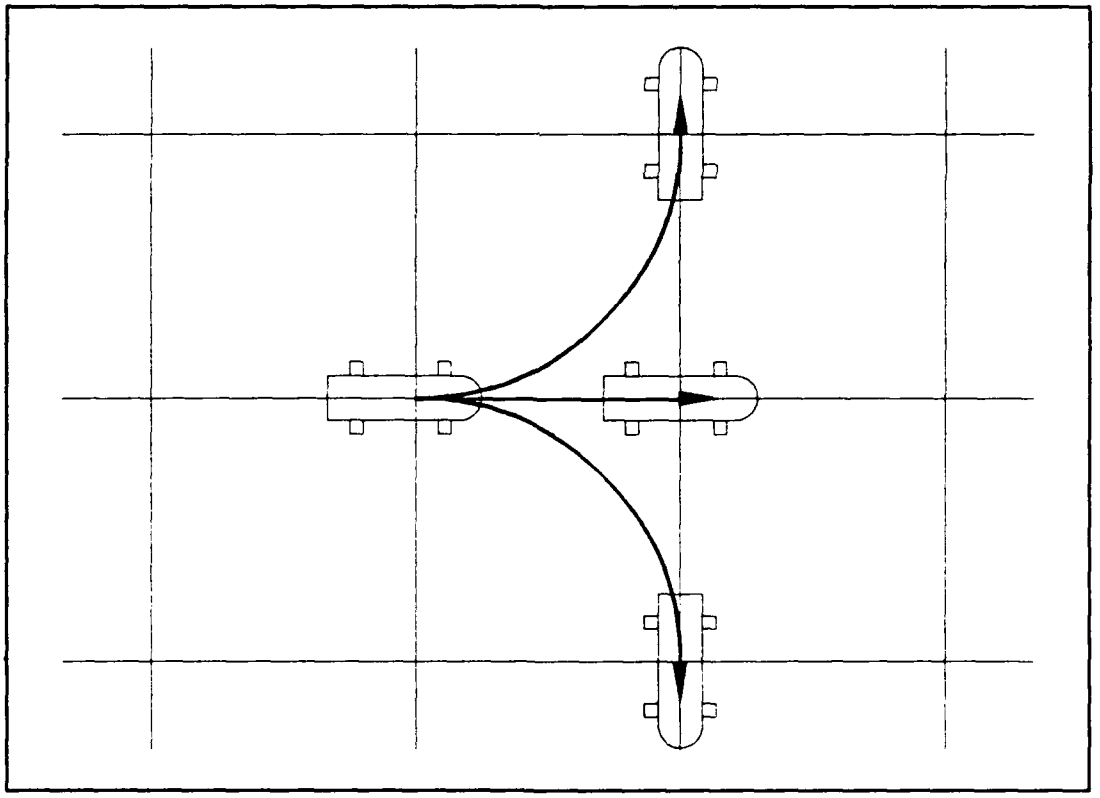

Figure 12 - Allowable Maneuvers

The basic size for the grid is established during sea-trials. Basic grid size is the maximum of the x and y values (advance and transfer) obtained as the vehicle executes a 90° course change under worst case conditions (most probably at low speed).

b. Output from Mission Planner

The output from the Mission Planner is a series of waypoints (x,y,z,v) which describe the optimal path and desired velocities to be achieved between the start and goal points. Note that this path is optimal only for the given set of boundary conditions. The Grid size is determined off-line based on the AUVs ability to execute a 90 degree course change under worst case conditions. For example, if the worst case calculations showed a change in x of 70 cm and a change in y of 90 cm in 4 sec. Also assume that the worst case expected current resulted in a set of 10 cm/sec. The maximum dimension would be 130 cm (90 cm plus 10 cm/sec for four seconds). The resulting grid would be 130 cm on a side. These waypoints are coupled with reference vehicle velocity to form a file of waypoint-velocity pair which is downloaded to AUV II. Since the Mission Planner has knowledge of possible obstacles in the AUVs world, the area between these waypoints is assumed to be obstacle free.

The Mission Planner can also be programmed to correct for expected set and drift encountered as the vehicle maneuvers. This capability will be especially beneficial when the AUV is operated in the bay or at sea. As an example, assume that the worst case measured values for advance and transfer are 100 cm and 80 cm

respectively, and that these values were obtained at a vehicle velocity of 0.5 meter/second. Using these values, the basic grid size would be 100 cm. If the expected set and drift was 0.1 meter/second (direction is irrelevant), then the correction is given by:

$$corr = \frac{\pi * \frac{Grid\_sz}{2}}{trial\_vel} \tag{4-1}$$

and the final grid size would be 130 cm (including a correction of 30 cm). The proposed x, y, and z positions that the Mission Planner outputs to Guidance are integral values of the final grid size.

2. Autopilot

The Autopilot receives desired waypoint, heading and velocity values from Guidance. These desired values are reference values which have been corrected based on current vehicle parameters [*RILING 90*]. The Autopilot outputs commands to the control surfaces, and main motors as it tries to achieve the desired attitude.

3. Navigation

The Navigation subsystem determines current vehicle position, heading, velocity and acceleration and provides this information to Guidance for comparison with reference waypoint-velocity information and deduced reference velocity and acceleration values [*FRIEND 89*].

D. OFF-LINE LOCAL PATH PLANNER

FIND_CS is the off-line program which performs the cubic spiral calculations for Local Path Planning in the Guidance subsystem. It is a modification of the cubic spiral calculation program developed by Kanayama for Yamabico-11 [*KANAYAMA 88a*].

The program takes as input a file consisting of the 27 possible combinations of x and y positions in the xy-plane. FIND_CS takes these 27 sets of data, creates two postures P_1 and P_2, determines the appropriate cubic spiral (or pair of cubic spirals) which joins these two postures, and writes the associated cubic spiral data to the output file. The position combinations are four sets of xy-pairs $\{(x_1,y_1),(x_2,y_2), (x_3,y_3)$ and $(x_4,y_4)\}$ which are used to calculate P_1 and P_2 as described below.

1. Reference Heading

Each posture is calculated using three of the pairs. P_1 is calculated using the first three xy-pairs $\{(x_1,y_1),(x_2,y_2)$, and $(x_3,y_3)\}$, and P_2 is calculated from the last three xy-pairs $\{(x_2,y_2),(x_3,y_3)$, and $(x_4,y_4)\}$. The xy-position for the posture is taken from the inner xy-pair of the triple (for P_1 the xy-position would be (x_2,y_2) and for P_2 it would be (x_3,y_3)). The reference heading for each posture is calculated in the same manner as was described earlier, where heading is the arc tangent of the difference between the previous and the next positions on the grid. As an example, if the input pairs are ((0,0)(1,1)(2,2)(2,3)), then the P_1 and P_2 positions would be (1,1) and (2,2) and the associated angles would be:

$$\theta_1 = \tan^{-1}\frac{(2 - 0)}{(2 - 0)} = 45° \qquad (4\text{-}2)$$

and

$$\theta_2 = \tan^{-1}\frac{(3 - 1)}{(2 - 1)} = 63.43° \qquad (4\text{-}3)$$

respectively.

2. Cubic Spiral Calculation

Once the postures have been determined, the next step is to determine the appropriate cubic spiral or cubic spirals to join the waypoints in the same manner as Yamabico-11, specifically:

(1) If P_1 and P_2 are symmetric, then find the cubic spiral which joins P_1 and P_2.

(2) If P_1 and P_2 are <u>not</u> symmetric, then find the split posture (Q) between P_1 and P_2 such that the sum of the cost for (P_1,Q) and (Q,P_2) is minimum.

(3) Find the cubic spiral joining P_1 and Q, and then do the same for the cubic spiral joining Q and P_2.

After the appropriate cubic spirals have been found the associated length (l) and curvature constant (α) values are written to the output file. The curvature constant is:

$$\alpha = \frac{6 * (\theta_2 - \theta_1)}{l^3} . \qquad (4\text{-}4)$$

Both of these terms and their importance to the local path-planning calculation are described fully in [*KANAYAMA 88a*].

E. ON-LINE LOCAL PATH PLANNER

The local path-planner must find a smooth path from waypoint to waypoint. In addition to waypoint information, the desired vehicle heading at each waypoint should also be included in the calculation, and can be inferred from waypoint information.

The desired vehicle heading at a waypoint is taken to be the arc-tangent of the two dimensional Cartesian difference between the previous waypoint (x_1, y_1) and the next waypoint (x_2, y_2):

$$\theta = \tan^{-1}\frac{(x_2-x_1)}{(y_2-y_1)} . \qquad (4\text{-}5)$$

Figure 13 illustrates this calculation in the case where the prior waypoint and the current waypoint have the same x grid values, and the following waypoint has x and y values one grid unit larger resulting in a desired heading of 27.5 degrees.

A posture (x,y,z,v,θ) is defined as a combination of the Cartesian coordinates of a point, a velocity, and a heading. Two postures are required as input to the local path-planner. The output from the local path-planner is a smooth "path" which satisfies the input conditions while minimizing total cost. A path (π) can be represented as a function of length (s) along the path where the tangent direction (θ) and curvature (κ) of π are:

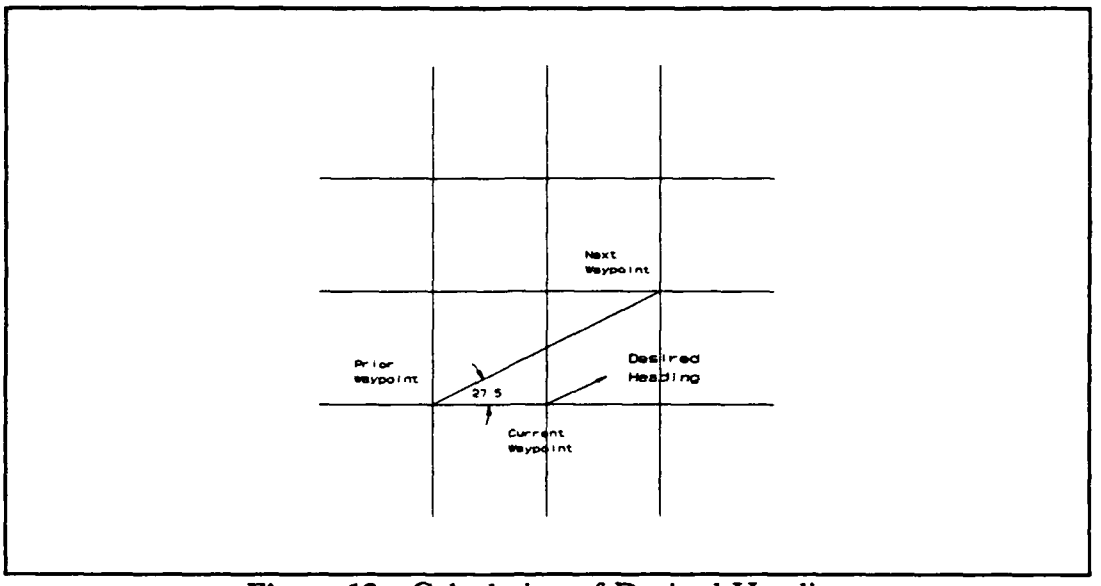

Figure 13 - Calculation of Desired Heading

$$\theta(s) = \tan^{-1}\left(\frac{\frac{dy}{ds}}{\frac{dx}{ds}}\right) \tag{4-6}$$

$$\kappa(s) = \frac{d\theta}{ds} \, . \tag{4-7}$$

1. Cardinal Heading Maneuvers

The vehicle is initially assumed to be heading in a cardinal direction (North, South, East or West). The 'Cardinal Heading' is calculated by the Guidance subsystem based on the first three waypoints. This calculation, and its effect on Guidance, is explained below.

With the stipulation that the vehicle can only turn left (L), go straight (C), or turn right (R), and *that the vehicle must end up on the grid*, all possible

combinations of vehicle maneuvers can be reduced to a set of twenty-seven curves, each of which can be identified by a triple (e.g. LLL) as specified in Table V. In Figure 14, the vehicle was initially heading East, then turned to the North (L), the West (L) and then to the South (L), for a combined (LLL) maneuver. The resultant cubic spiral is a quarter circle to the left.

Table V - Possible Combinations of Maneuvers

LLL	CLL	RLL
LLC	CLC	RLC
LLR	CLR	RLR
LCL	CCL	RCL
LCC	CCC	RCC
LCR	CCR	RCR
LRL	CRL	RRL
LRC	CRC	RRC
LRR	CRR	RRR

Any maneuvers which require more than these three changes in cardinal heading can be simulated through some combination of these twenty-seven triples.

Since there are only twenty-seven predefined possible maneuvers, the cubic spiral information is precalculated using Kanayama's method [*KANAYAMA 88a*], and placed into a lookup table, resulting in a tremendous increase in calculation speed.

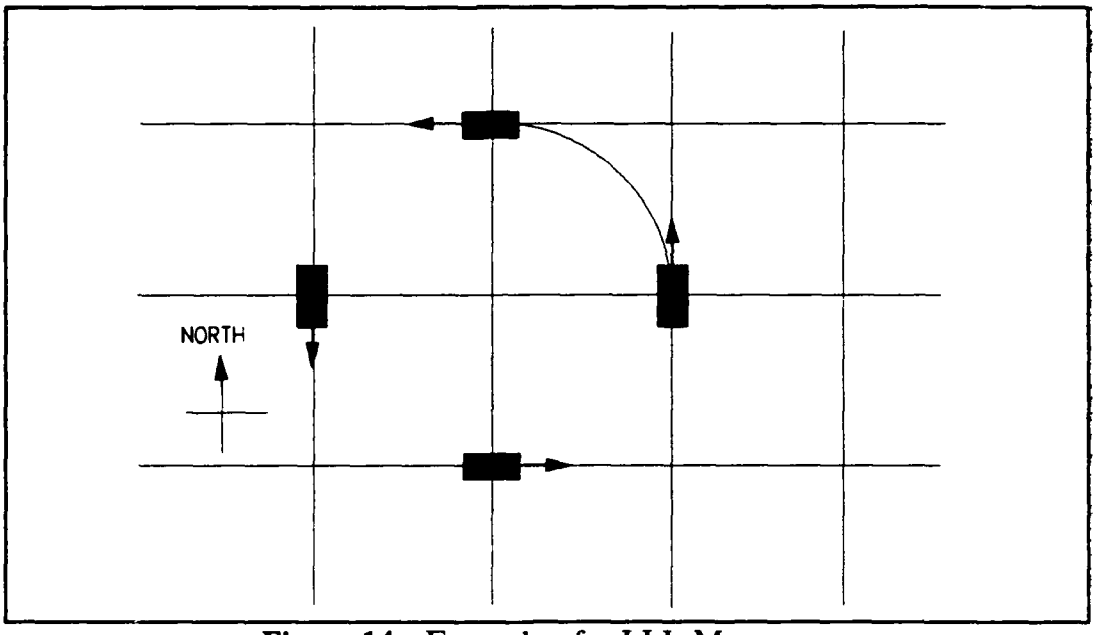

Figure 14 - Example of a LLL Maneuver

2. Theory of Operation

The objective of the LPP is to provide appropriate data to the Stepper which calculates the next desired posture to follow between waypoints. The Stepper requires information on the length of a curve (d) and curve deflection (α), where d is the Euclidean distance between the waypoints and α is the angle between the desired vehicle heading at each of the waypoints. This pair (d,α) is called a curve set.

On Yamabico-11, the local path planner must perform curve set calculations on-line, since the waypoint orientation is unknown [*KANAYAMA 89a*]. For AUV-II the task is much simpler. The local path calculations are expedited by using a stored curve database which contains all allowable combinations of vehicle

maneuvers. By eliminating on-board calculations of curve parameters, the real-time response of the LPP is dramatically improved.

Because of the simplified set of twenty-seven possible combinations of waypoints (as indicated in Table V) achievable by the AUV, the curve set information is calculated by the off-line local path planner and is placed into a lookup table for use by the on-line local path-planner. The off-line program utilizes path curvature and the derivative of path curvature as cost functions as follows:

$$cost = \alpha^2 * D(\alpha)^3 . \tag{4-8}$$

As specified by Kanayama, there is only one simple path with a given curve set (d,α)[*KANAYAMA 88a*]. If a set of waypoints (x_1,y_1) (x_2,y_2) *is* symmetric, then the desired curve set consists of a single cubic spiral. The desired solution to a set of waypoints *is symmetric if and only if*

$$\theta_1 - \beta = \beta - \theta_2 \tag{4-9}$$

where

$$\beta = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right). \tag{4-10}$$

If the waypoint pair is not symmetric then two curves which satisfy the minimum cost function are required, resulting in two curve sets. The lookup table utilized by the local path-planner has two curve sets for each of the possible combinations, with the second curve set placed to zero if the waypoint pair is

symmetric. These lookup table are size-free and can be scaled to any desired grid size.

Once a set of curve information has been retrieved, **STEP_SPIRAL** is used to calculate a series of intermediate reference postures which are used by the Tracking Controller. A reference posture is defined as a quadruple of reference x-position, y-position, depth, speed and heading (x,y,z,v,θ).

3. Waypoint Processing

As waypoints are passed to the local path-planner, a new vessel pseudo-heading is computed based on the past way-point (x_0,y_0) and the current way-point's successor (x_2,y_2). This pseudo-heading is used to calculate relative direction change and the new vessel cardinal heading. The new pseudo-heading (θ_1) is given by:

$$\theta_1 = \tan^{-1}\left(\frac{y_2 - y_0}{x_2 - x_0}\right) - \psi_0 \qquad \textbf{(4-11)}$$

where ψ_0 is the cardinal heading which existed prior to the maneuver. Using Figure 14 as an example again, assume the vessel was initialially heading East, the pseudo-heading to the next waypoint would be -90.0°, resulting in a relative direction change of (L), and a new cardinal heading of North. This relative dirction change is combined with the last two relative direction changes to form a new triple. The triple is used as the table entry for the table lookup of the appropriate cubic spiral curve information.

The desired vehicle heading calculation at the start of the mission is somewhat more complex. Since there is no previous waypoint, the initial reference

heading must be calculated based on the following two waypoints. This is accomplished by determining whether the change in the x and y directions are positive, negative or zero. Two pairs (sign-x sign-y) are calculated and reference vehicle heading and dummy values for the previous position (x_0,y_0) are chosen based on these pairs.

4. Reference Value Calculation

Reference values are dead-reckoned from the point where the mission originated. The length of a "step" from one reference posture to the next is current vessel velocity multiplied by execution time interval.

Let d_0 denote the distance of a point from the start of the path, and s the size of an incremental change in d_0. Then the distance after an incremental change becomes:

$$d_1 = d_0 + s \tag{4-12}$$

and the curvature κ can be determined from:

$$\kappa = A * d_1 * (l - d_1) \tag{4-13}$$

where l is the total length of the cubic spiral. The change in the reference heading ($\Delta\theta_r$) is then:

$$\Delta\theta_r = \kappa * s \tag{4-14}$$

and the new reference heading is:

$$\theta_r = \theta_r + \Delta\theta \ . \tag{4-15}$$

The new reference values for x and y (x_r,y_r) are calculated from simple trigonometric relationships:

$$x_r = \cos(\theta_r) * s \tag{4-16}$$

$$y_r = \sin(\theta_r) * s \ . \tag{4-17}$$

F. TRACKING CONTROLLER

The purpose of the Tracking Controller is to provide desired postures to the Autopilot. The Tracking Controller for AUV II is modelled after the system utilized in Yamabico-11 by Kanayama because of its proven capability, robustness and ease of implementation [*KANAYAMA 90*].

1. Basic Operation

The Tracking Controller receives as input reference postures and velocities from Step_Spiral which define the path that the vessel should follow, along with actual position and velocities from Navigation which show the actual vehicle path. An error value calculation is performed and applied to the reference position to get the desired values of heading, velocity and position to provide to the Autopilot.

The AUV in the world possesses six degrees of freedom in positioning which are represented by a *posture* (x,y,z,θ,ϕ,ψ), which is also a function of time. The

set of positions (x,y,z) that the vessel traverses is called a *path*, and it can easily be shown that heading is actually:

$$\dot{\theta} = \tan^{-1}(\frac{\dot{y}}{\dot{x}}) \tag{4-18}$$

(where the dot is defined to be the derivative with respect to time).

The velocity of the vehicle in three-dimensional space can be specified as: a triple:

$$q=(v,\omega,\psi) \tag{4-19}$$

where v is the AUV speed, ω is rotational velocity in the x-y plane and ψ is the rotational velocity in the x-z plane (we are neglecting vehicle roll).

The relationship between the posture p and the velocities q is given by:

$$q = (v,\omega,\dot{\phi}) = (\dot{x}\cos\theta\cos\phi + \dot{y}\cos\theta\sin\phi,\ \dot{\theta},\ \dot{\phi}) \tag{4-20}$$

where θ is the current vehicle heading and ϕ is the current vehicle pitch.

2. Error Value Calculation

The error value determination method used in AUV II is identical to that used in Yamabico-11. The error values are the difference between reference and actual positions scaled by actual heading and are calculated from:

$$x_e = (x_r - x_c) * \cos(\theta_c) + (y_r - y_c) * \sin(\theta_c) \tag{4-21}$$

$$y_e = (x_c - x_r) * \sin(\theta_c) + (y_r - y_c) * \cos(\theta_c) \tag{4-22}$$

$$\theta_e = \theta_r - \theta_c\ . \tag{4-23}$$

3. Desired Value Calculation

As the vehicle proceeds from waypoint to waypoint, it is constantly comparing actual posture (P_c) to reference posture (P_r). There are three possible outcomes from this calculation:

- The vehicle is ahead of the reference posture (x_e positive).
- The vehicle is at the reference posture (x_e approximately zero).
- The vessel is behind the reference posture (x_e negative).

If x_e is positive, a new reference posture is calculated by the on-board LPP, desired velocity is decreased, and new x_r and y_r values are calculated and passed to Autopilot. If x_e is nearly zero, a new reference posture is calculated by the on-board LPP, v_d remains unchanged and a new desired posture is calculated and passed on to Autopilot. If the AUV is lagging behind (x_e negative), then the reference posture remains the same, x_d and y_d are recalculated, and v_d is increased and passed on to Autopilot.

For AUV II, the desired x and y values are calculated using the same equations developed for Yamabico-11, and are then scaled to the current vessel grid size:

$$x_d = (x_r + x_e) * GRID\_SZ \tag{4-24}$$

$$y_d = (y_r + y_e) * GRID\_SZ \tag{4-25}$$

where the error values and reference values are calculated as above.

Desired velocity determination is also the same as for Yamabico-11:

$$v_d = (K_x * x_e) + (v_c * \theta_e) . \quad (4\text{-}26)$$

The calculation of desired heading is somewhat more complex. Since ω is the derivative of heading, the desired change in heading can be obtained by:

$$\Delta\theta = \omega_c + v_c * (K_y * y_e + K_\theta * \theta_e) \quad (4\text{-}27)$$

where ω_c and v_c are the actual angular velocity and linear velocity supplied by the Navigation subsystem. This change in heading ($\Delta\theta$) is added to the old desired heading to determine the new desired heading. K_x, K_y and K_θ are integration constants whose values are calculated using the methods specified by Kanayama [*KANAYAMA 90*]. For the purposes of validating the Guidance Subsystem software without a vessel, these constants were calculated to be: K_x = 10/sec, K_y = 0.0064/cm^2, and K_θ = 0.16/cm. The final values for these constants must be determined and verified by experimentation when the vehicle is in the water.

4. Look-Ahead Limitation

The Tracking Controller is designed to "drag" AUV II along from desired posture to desired posture. It is desirable for the next desired posture to be just far enough ahead of the current posture that the vehicle will reach the desired posture after a certain number of cycles of Autopilot operation. For the Tracking Controller in AUV II the Autopilot is allowed ten cycles to achieve the desired posture.

The selection of the factor of ten difference in the timing between the Tracking Controller and the Autopilot is quite common in AUV control [*BLIDBERG 90, ALBUS 90*], and provides for adequate separation of the various layers of control.

If the vehicle is unable to achieve the desired position in ten cycles, then a decision must be made either to calculate a new desired position, or to continue to the old desired position. This is accomplished by comparing the size of x_e to a reference maximum value. As long as x_e is negative and less than the maximum, a new reference value is calculated. If x_e becomes greater than the maximum, the old reference posture is passed to the Autopilot.

If the vehicle gets ahead of the desired posture, a new posture is calculated, and in addition, the vehicle velocity is decreased by an amount proportionate to the size of x_e.

G. TESTING RESULTS FOR TWO-DIMENSIONAL GUIDANCE

The two-dimensional Guidance subsystem source code was compiled and installed on the GESPAC Vehicle Control Computer. The output of the Guidance subsystem was written to a file along with the waypoints and the smooth cubic spiral curves between waypoints. Since there is neither an operational navigator nor a valid simulation model for AUV II, the last desired parameters were fed back as actual vehicle parameters to Guidance. This artificiality is not a major concern, because the Tracking Controller system has already been operationally tested in Yamabico-11 [*KANAYAMA 89*].

Several files of waypoints generated by the Mission Planner were downloaded to the GESPAC via an RS-232 serial port, and the missions were run to completion. All missions resulted in adequate vehicle performance.

The results of one such mission are shown in Figure 15. The path followed by Step_Spiral is the solid line. The waypoints provided by the Mission Planner are the diamonds. The desired postures fed to the Autopilot are the crosses. As can be seen from this drawing, the Tracking Controller was able to "drag" the simulated vehicle along with no discernable error.

H. THREE-DIMENSIONAL CALCULATIONS

As was previously mentioned, the Guidance subsystem described thus far only performs local path-planning and Tracking Control in two-dimensions. The addition of depth brought new and interesting problems.

1. Depth Descriptors

Since the vehicle was operating on a three-dimensional grid, it was possible to create a set of curve descriptors which would apply to depth. The vehicle isrestricted to moving ahead, up or down just one grid square at a time (U,S,D). These three possiblities are applied in the same manner as for the two-dimensional case, and once again result in a maximum of 27 possible combinations of cubic spiral curves.

The primary problem here is that unless the z grid dimension is the same as the x and y dimensions (resulting in a cubic grid pattern), then the depth curve

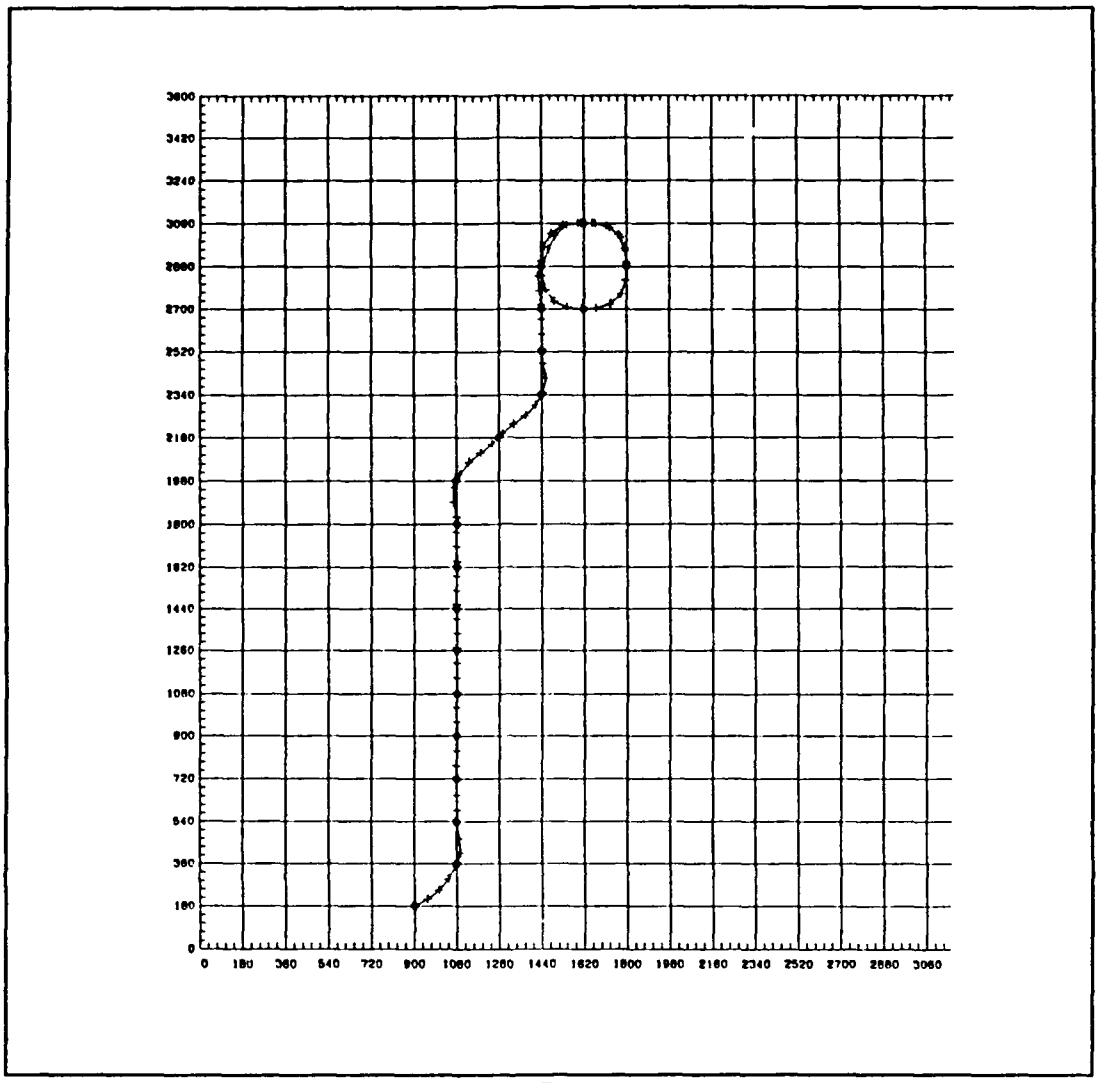

Figure 15 - Guidance Subsystem Performance

descriptors will have to be chosen based on the scale difference. As an example, if the x/y dimension is 100 cm and the z dimension is 10 cm, then the depth curves must be calculated based on this relationship. The program FIND_D (see Appendix H for three-dimensional source code) performs these calculations. It requires an input file showing the relationship between the waypoints, and outputs an appropriate set of cubic spiral curve descriptors.

2. Modifications to On-Line Local Path Planner

The vehicle must travel from waypoint to waypoint on the three-dimensional grid, and it must arrive at the x-y waypoint coordinate at the same time as it arrives at the z waypoint coordinate. This poses the hardest problem in achieving three-dimensional guidance. There must be some time synchronization between stepping through the x-y curve set and the z curve set.

The total distance to travel in each dimension is known (it is the sum of the lengths of the individual cubic spirals (l)) and the number of steps in each dimension must remain equal. From this it is easy to see that to travel the same number of steps, and yet go a different distance, the step sizes must be different. Therefore, the z step size is calculated from:

$$z\_step = xy\_step * \frac{(xy.len1 + xy.len2)}{(z.len1 + z.len2)} . \qquad (4\text{-}23)$$

By keeping the number of steps the same, the time spent in each curve will be identical, and the vessel will complete both curves at the same time.

VI. CONCLUSIONS AND RECOMMENDATIONS

This chapter summarizes the contributions which have been made to AUV research in general, and specifically to the NPS AUV II. It also provides recommendations for future research in AUV II guidance and control.

A. CONTRIBUTIONS

1. Vehicle Control Computer System

The necessary computer equipment was identified, procured and configured to ensure that all identified data processing and data translation requirements at the Vehicle Control level were met or exceeded. In addition, the chosen VCC allows for data translation capability upgrades, and for expansion to multiple processors at some future date.

2. Guidance Subsystem

A functional Guidance subsystem has been implemented and tested on the Vehicle control computer. This subsystem calculates a smooth path from waypoint to waypoint while minimizing vessel jerk. The Tracking Controller in this subsystem provides all necessary information to the Autopilot at a user specified frequency.

The Tracking Controller module is robust enough to be used to provide vehicle control commands without an Autopilot.

3. Vehicle Firmware

All data translation firmware necessary for the Vehicle Control Computer has been designed, implemented and tested. This software should be able to function totally unchanged for the life of the project.

In addition real-time software has been developed and tested for the main motors and hydrodynamic control surfaces. The 'lookup table' function used to control these effectors and the algorithm used in creating the lookup tables for these devices can be applied to all other vehicle control mechanisms.

This combination of firmware provides all software necessary for initial in-water testing of AUV II.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Real-Time scheduling

The single most important decision facing the AUV II real-time computer systems is the design and implementation of a multi-tasking scheduling scheme. The OS-9 operating system is designed to allow the use of several possible methods of multi-tasking [*DIBBLE 88*]. The event driven control method used by ISE [*ZHENG 90*] is another viable option, as is the real-time pre-emptive scheduling methods being pursued by MIT-Sea Grant [*BELLINGHAM 90*]. This scheduling decision will drive the design and implementation of all other support modules.

2. Guidance Subsystem

The Guidance subsystem should be linked to the off-hull Mission Planner to allow for accurate real-time simulation of mission plans. This integration, combined with the integration of Autopilot and Navigator, and an improved vehicle dynamic model, will allow missions to be run in real-time, which would greatly enhance the credibility of the vehicle simulatons.

The Guidance subsystem must be interfaced to the Autopilot when the Autopilot has been operationally tested on the Vehicle Control Computer. Further testing of the Guidance subsystem should also be accomplished using simulated vehicle dynamic inputs to verify the robustness of the Tracking Controller.

3. Communication

There is an urgent need to establish real-time communications between the GESPAC and the GRiDCASE computers. Waypoints must be sent from the GRiDCASE to the GESPAC, and vehicle attitude and positional information must be sent from the GESPAC to the GRiDCASE to allow for Mission Replanning. In order to facilitate initial vehicle testing, an RS-232 link should be connected between the GESPAC and the pierside GRiDCASE/PC. The removable fiber-optic data link would be the best choice for this communications connection.

4. Vehicle Firmware

Routines must be written which will ensure that equipment in AUV II is powered up in the proper order, and which will ensure proper operation of the

effectors prior to launch. In addition, firmware is required for the inertial sensor suite, for the health and well-being sensors, and for the tunnel thrusters.

APPENDIX A. GRiD TIMED INTERRUPT SOURCE CODE

```
/*
        Program:  TEST_TSR.C
        Purpose:  timed interrupt test program for GRiDCASE 1535 EXP
        Author:   LT M.J. CLOUTIER

        TEST_TSR does the following:

        1. Using vector 66h, checks whether the clock is already installed
        2. Saves original clock vector
        3. Sets up new clock vector (Install_Autopilot)
        4. Does a "do nothing loop" to evaluate the TSR function

        (NOTE:  This code is an extensive modification of a timer routine in the DOS
                        Programmers Reference by DETTMAN)
*/

#include <stdio.h>
#include <dos.h>
#include <conio.h>

/* constants */
#define     PGMSIZE 3000

/* define base address for video display */
#define         MONOBASE    0xb000
#define         COLORBASE   0xb800
#define         EGABASE     0xa000

/* interrupt vectors for BIOS and DOS */
#define         AUTOPILOT    0x66
#define         TIMER        0x1c
#define         VIDEO        0x10

void interrupt (*orig_clock)(void);  /* original clock vector */
void interrupt autopilot(void);/* declare autopilot() */
void Install_Autopilot(void);
void Remove_Autopilot(void);
```

```
int   tick_count = 0;          /* clock tick counter */
int   sp;                  /* stack pointer */
int   ss;                  /* stack segment */
int   count = 0;
char  buf[80] = "<--------";
char  far *auvptr;

main()
{
      union  REGS   regs;
      int           mode;
      int           j;

      Install_Autopilot();
      for(j=0 ;j < 20;j++){
           printf("Count=%d\n",count);
           delay(50);
      }
      Remove_Autopilot();
}

void  Install_Autopilot(void)
/*  Set up the Autopilot interrupt handler, saving machines clock program to
orig_clock */
{
      int        mode;

      /* sets auvptr for appropriate screen mode  */
      mode = getmode();
      printf("Display mode is %d\n",mode);
      if (mode==7)
           auvptr = MK_FP(MONOBASE,3856);
      else if (mode==3)
           auvptr = MK_FP(COLORBASE,3856);
      else
           auvptr = MK_FP(EGABASE,3856);
      orig_clock = getvect(TIMER);
      setvect(TIMER,autopilot);
} /* Install_Autopilot */

void  Remove_Autopilot(void)
/* Removes Autopilot interrupt handler, restores machines clock program */
{
```

```
        setvect(TIMER,orig_clock);
} /* Remove_Autopilot */

void interrupt autopilot(void)
/*      Autopilot interrupt handler ... called by the timer interrupt
        18.2 times per second.  Every 5th second add one more tick
        before advancing to keep average time correct
*/
{
        static int INSIDE_AUTOPILOT = 0;
        static unsigned long Count = 0;

        /* call the original interrupt first */
        (*orig_clock)();

        /* Advance the tick counter */
        tick_count++;

        /* Every 20th tick write Autopilot string */
        if (!INSIDE_AUTOPILOT && !(tick_count ^ 0x0a)){
                tick_count = 0;
                INSIDE_AUTOPILOT = 1;
                disable();
                sp = _SP;
                ss = _SS;
                _SS = _CS;
                _SP = PGMSIZE;
                enable();

                itoa(++count,&buf[8],10);
                displaystr(buf);
                for (;Count < 400000;Count++);

                disable();
                _SP = sp;
                _SS = ss;
                enable();
                Count = 0;
                INSIDE_AUTOPILOT = 0;
        }
} /* autopilot */

displaystr(str)
```

```
/* Display the string at the clkptr position on the screen */
char        *str;
{
      char       far   *ptr;

      ptr = auvptr;
      while(*str){
           *ptr++ = *str++;
           ptr++;
      }
} /*displaystr */

getmode()
/* Returns video display mode of the system  */
{
      union     REGS    regs;

      regs.h.ah = 0x0f;
      int86(VIDEO,&regs,&regs);
      return(regs.h.al);
} /* getmode */
```

APPENDIX B. DATA TRANSLATION SOURCE CODE

```
/*
 *    Program --- ALL_ADDA.C
 *    Purpose --- Data Translation routines for GESPAC DAC2B, ADA1 and
ADC2C          *          cards for AUV II
 *    Author  --- LT M.J. CLOUTIER
 *    (Some of these routines are from data translation routines used in the MIT Sea
Sprite)
 */

#define DAC_ADDR        0xFFF00000
#define DAC0_MSB        0x0
#define DAC1_MSB        0x4
#define DAC2_MSB        0x8
#define DAC3_MSB        0xc
#define DAC_LSB_OFFSET  0x2

#define ADC_ADDR        (DAC_ADDR + 0x11)
#define ADC_MSB         0x0
#define ADC_LSB         0x2
#define ADC_CMD_REG     0x4
#define ADC_STATUS_REG  0x4
#define ADC_BUSY        0x4

#define DAC2B_ADDR      0xFFF00040
#define DAC4_MSB        0x10
#define DAC5_MSB        0x14
#define DAC6_MSB        0x18
#define DAC7_MSB        0x1c

#define ADC2_ADDR       0xFFF00020
#define ADC2_CH_GAIN    0x0
#define ADC2_STATUS_REG 0x2
#define ADC2_DATA       0x1
#define ADC2_CMD_REG    0x2

unsigned short *adc2_a     = ADC2_ADDR;
unsigned char  *dac2b_a    = DAC2B_ADDR;
unsigned char  *ada1_dac_a = DAC_ADDR;
unsigned char  *ada1_adc_a = ADC_ADDR;
```

```
/*********************************************************
 *  ada1_adc(n) -- reads adc channel 'n' (channels 0-15)
 *********************************************************/
int ada1_adc(n)
int n;
{
   int val;

   ada1_adc_a[ADC_CMD_REG] = n;
   while (ada1_adc_a[ADC_STATUS_REG] > 20);    /* wait for data */
   val = ada1_adc_a[ADC_MSB] << 2;
   val += ada1_adc_a[ADC_LSB] >> 6;
   return (val);
} /* ada1_adc */

/*********************************************************
 *  adc2_adc(n) -- reads adc channel 'n' (channels 0-15)
 *                 with gain 'g'(0 to F => 0 - 1024)
 *********************************************************/
int adc2_adc(n,g)
int n,g;
{
   int val;

   adc2_a[ADC2_CH_GAIN] = (n << 4) | g; /* set c&g, start conv */
   while (adc2_a[ADC2_STATUS_REG] != 0); /* wait for ready */
   val = adc2_a[ADC2_DATA];
   return (val);
}/* adc2_adc */

/*********************************************************
 *  ada1_dac(s,ch) -- writes signal 's' to ada1 dac channel 'ch'
 *              (allowable channels 0-3)
 *********************************************************/
void ada1_dac(s,ch)
int s,ch;
{
   ch = ch << 2;                          /* offset for G-96 addressing */
   ada1_dac_a[ch] = s >> 2;                 /* write upper 8 bits to MSB*/
   ada1_dac_a[ch + DAC_LSB_OFFSET] = s << 6; /* write lower 2 bits B3,B2 */
   return;
}/* ada1_dac */
```

```
/*********************************************************
 *  dac2_dac(s,ch) -- writes signal 's' to dac2 channel 'ch'
 *                    (allowable channels 0-7)
 *********************************************************/
void dac2_dac(s,ch)
int s,ch;
{
   ch = ch << 2;                          /* offset for G-96 addressing */
   dac2b_a[ch] = s >> 4;                    /* write upper 8 bits to MSB*/
   dac2b_a[ch + DAC_LSB_OFFSET] = s << 4;  /* write lower 4 bits B3-B0 */
   return;
}/* dac2_dac */

/*********************************************************
 * read_ada1_dac(ch) -- read output from dac
 *********************************************************/
int read_ada1_dac(ch)
int ch;
{
   int s;

   ch = ch << 1;
   s = ada1_dac_a[ch] << 2;
   s = s + (ada1_dac_a[ch + DAC_LSB_OFFSET] >> 6);
   return (s);
}/* read_ada1_dac */

/*********************************************************
 *  ada1_2_ada1(n1,n2) -- signal from ada1 adc(n1) written
 *                        to ada1 dac(n2)
 *********************************************************/
void ada1_2_ada1(n1,n2)
int n1,n2;
{
   int val;

   n2 = n2 << 2;
   ada1_adc_a[ADC_CMD_REG] = n1;
   while (ada1_adc_a[ADC_STATUS_REG] > 20);  /* wait for data */
   ada1_dac_a[n2] = ada1_adc_a[ADC_MSB];
   ada1_dac_a[n2 + DAC_LSB_OFFSET] = ada1_adc_a[ADC_LSB];
}/* ada1_2_ada1 */
```

```
/*********************************************************
 *  adc2_2_ada1(n1,n2) -- signal from ada1 adc(n1) written
 *                        to ada1 dac(n2)
 *********************************************************/
void adc2_2_ada1(n1,g,n2)
int n1,g,n2;
{
   unsigned short val;

   n2 = n2 << 2;
   adc2_a[ADC2_CH_GAIN] = (n1 << 4) | g;
   while ((adc2_a[ADC2_STATUS_REG]  & 0x7) != 0);  /* wait for data */
   val = adc2_a[ADC2_DATA];
   ada1_dac_a[n2] = val >> 4;
   ada1_dac_a[n2 + DAC_LSB_OFFSET] = val << 4;
}/* adc2_2_ada1 */

/*********************************************************
 *  ada1_2_dac2(n1,n2) -- signal from ada adc(n1) written
 *                        to dac ch(n2)
 *********************************************************/
void ada1_2_dac2(n1,n2)
int n1,n2;
{
   n2 = n2 << 2;
   ada1_adc_a[ADC_CMD_REG] = n1;
   while (ada1_adc_a[ADC_STATUS_REG] > 20);  /* wait for data */
   dac2b_a[n2] = ada1_adc_a[ADC_MSB];
   dac2b_a[n2 + DAC_LSB_OFFSET] = ada1_adc_a[ADC_LSB];
}/* ada1_2_dac2 */

/*********************************************************
 *  adc2_2_dac2(n1,n2) -- signal from ada adc(n1) written
 *                        to dac ch(n2)
 *********************************************************/
void adc2_2_dac2(n1,g,n2)
int n1,g,n2;
{
   unsigned short val;

   n2 = n2 << 2;
   adc2_a[ADC2_CH_GAIN] = (n1 << 4) | g;
   while ((adc2_a[ADC2_STATUS_REG] & 0x7) != 0);  /* wait for data */
```

```
    val = adc2_a[ADC2_DATA];
    dac2b_a[n2] = val >> 4;
    dac2b_a[n2 + DAC_LSB_OFFSET] = val << 4;
}/* adc2_2_dac2 */

main (argc,argv)
int argc;
char *argv[];
{
    int val;

    /* This program reads from the ADA1 or ADC2B ADC and writes to the DAC2B
DAC        * to test the data conversion routines
     */
    printf("ALL A2D && D2A: testing\n");
    while (1){
/*
        ada1_2_dac2(0,0);
        ada1_2_dac2(1,1);
        ada1_2_dac2(2,2);
        ada1_2_dac2(3,3);
        ada1_2_dac2(4,4);
        ada1_2_dac2(5,5);
        ada1_2_dac2(6,6);
        ada1_2_dac2(7,7);
*/
        adc2_2_dac2(0,0,0);
        adc2_2_dac2(1,0,1);
        adc2_2_dac2(2,0,2);
        adc2_2_dac2(3,0,3);
        adc2_2_dac2(4,0,4);
        adc2_2_dac2(5,0,5);
        adc2_2_dac2(6,0,6);
        adc2_2_dac2(7,0,7);
        adc2_2_ada1(8,0,0);
        adc2_2_ada1(9,0,1);
        adc2_2_ada1(10,0,2);
        adc2_2_ada1(11,0,3);
        adc2_2_ada1(12,0,4);
    }
} /* ALL_ADDA */
```

APPENDIX C. SERVO CONTROL CODE

```
/*
 *     Program - CALC_SVO.C
 *     Purpose - perform servo lookup table calculations for AUV II
 *     Author -- LT M.J. CLOUTIER
 */

#include <math.h>
#include <stdio.h>

FILE  *outfp;

void calc_servo_val(angle)
double angle;
{
      double voltage;

      angle += 45.0; /* normalizes angle to 45 */
      if (angle < 15.0)
            voltage = (angle*0.086333)+0.27;
      else if (angle < 30.0)
            voltage = ((angle - 15.0)*0.08466) + 1.565;
      else if (angle < 45.0)
            voltage = ((angle - 30.0)*0.068) + 2.835;
      else if (angle < 60.0)
            voltage = ((angle - 45.0)*0.055) + 3.855;
      else if (angle <= 75.0)
            voltage = ((angle - 60.0)*0.04666) + 4.68;
      else if (angle <= 90.0)
            voltage = ((angle - 75.0)*0.04266) + 5.38;
      else {
            printf("Invalid angle: %4.2f\n",angle - 45.0);
            exit(-1);
      }
      fprintf(outfp,"\t%d,\n",(int)(102.4 * voltage));
} /* calc_servo_ang */

main (argc,argv)
int argc;
```

```
char *argv[];
{
        double ang;

        if ((outfp = fopen("servo.dat","w")) == 0){
                printf("Unable to open 'svo.dat'\n");
                exit(-1);
        }

        fprintf(outfp,"int svo_ang[91] = {\n");
        for (ang = -45.0; ang <= 45.0; ang += 1.0){
                calc_servo_val(ang);
        }
        fprintf(outfp,"};\n");
        fclose(outfp);
} /* CALC_SVO */

/*
 *      S.DAT - servo lookup table for AUV II
 *      Author - LT M.J. CLOUTIER
 */

int svo_ang[91] = {
        27,36,45,54,63,71,80,89,98,107,116,124,133,142,151,160,168,
        177,186,194,203,212,220,229,238,246,255,264,272,281,290,297,
        304,311,318,325,332,339,346,352,359,366,373,380,387,394,400,
        406,411,417,422,428,434,439,445,451,456,462,467,473,479,484,
        488,493,498,503,507,512,517,522,527,531,536,541,546,550,555,
        559,564,568,572,577,581,585,590,594,598,603,607,612,616
};

/*
 *      Program --- TEST_SVO.C
 *      Purpose --- servo test program for AUV II
 *      Author ---- LT M.J. CLOUTIER
 */

#include <stdio.h>
#include <math.h>
#include "s.dat"

#define DAC_ADDR        0xFFF00000
#define DAC_LSB_OFFSET  0x2
```

```
#define BOW             0

unsigned char *dac_a = DAC_ADDR;

/*********************************************************
 *  dac(s,ch) -- writes signal 's' to ada1 dac channel 'ch'
 *               (allowable channels 0-3)
 *********************************************************/
void dac(s,ch)
int s,ch;
{
   ch = ch << 2;                          /* offset for G-96 addressing */
   dac_a[ch] = s >> 2;                     /* write upper 8 bits to MSB*/
   dac_a[ch + DAC_LSB_OFFSET] = s << 6;    /* write lower 2 bits B3,B2 */
   return;
} /* dac */


void send_servo(surface,angle)
/*
 * This function sends the desired ANGLE to the specified control SURFACE
 * The angle is first normalized to (-45 to 45), then correction is applied
 * for the non-linearity in the servo control module
 */
int surface;
double angle;
{
      double voltage;
      if ((angle < -45.0) || (angle > 45.0)){
           printf("Angle out of range: %4.2f\n",angle);
      } else {
           dac(svo_ang[(int)(angle + 45.0)],surface);
      }
} /* send_servo */

main ()
{
      double ang;
      long j;

      ang = 0.0;
      while ((ang <= 45.0) && (ang >= -45.0)){
           printf("Enter desired angle from -45 to +45 degrees (999 to quit)\n");
```

```
            scanf("%lf",&ang);
            send_servo(BOW,ang);
        }
        send_servo(BOW,0.0);
    } /* TEST_SVO.C */
```

APPENDIX D. MAIN MOTOR CONTROL CODE

```
/*
 *    Program - CALC_MN.C
 *    Purpose - perform main motor lookup table calculations for AUV II
 *    Author -- LT M.J. CLOUTIER
 */

#include <math.h>
#include <stdio.h>

FILE  *outfp;

void calc_rpm(rpm)
double rpm;
{
      double voltage;

      if (rpm < -700){
           printf("Invalid RPM:  %4.2f\n",rpm);
           exit(-1);
      }
      else if (rpm < -600)
           voltage = ((2.73 - 2.01)/100*(rpm+700)) + 2.01;
      else if (rpm < -500)
           voltage = ((3.25 - 2.73)/100*(rpm+600)) + 2.73;
      else if (rpm < -400)
           voltage = ((3.74 - 3.25)/100*(rpm+500)) + 3.25;
      else if (rpm < -300)
           voltage = ((4.14 - 3.74)/100*(rpm+400)) + 3.74;
      else if (rpm < -200)
           voltage = ((4.45 - 4.14)/100*(rpm+300)) + 4.14;
      else if (rpm < -100)
           voltage = ((4.67 - 4.45)/100*(rpm+200)) + 4.45;
      else if (rpm < 0)
           voltage = ((5.0 - 4.67)/100*(rpm+100)) + 4.67;
      else if (rpm < 100)
           voltage = ((5.32 - 5.0)/100*rpm) + 5.0;
      else if (rpm < 200)
           voltage = ((5.56 - 5.32)/100*(rpm-100)) + 5.32;
```

```
        else if (rpm < 300)
            voltage = ((5.90 - 5.56)/100*(rpm-200)) + 5.56;
        else if (rpm < 400)
            voltage = ((6.39 - 5.90)/100*(rpm-300)) + 5.90;
        else if (rpm < 500)
            voltage = ((6.90 - 6.39)/100*(rpm-400)) + 6.39;
        else if (rpm < 600)
            voltage = ((7.45 - 6.90)/100*(rpm-500)) + 6.90;
        else if (rpm <= 700)
            voltage = ((8.28 - 7.45)/100*(rpm-600)) + 7.45;
        else {
            printf("Invalid rpm: %4.2f\n",rpm);
            exit(-1);
        }
        fprintf(outfp,"\t%d,\n",(int)(102.4 * voltage));
} /* calc_rpm */

main (argc,argv)
int argc;
char *argv[];
{
        double rpm;

        if ((outfp = fopen("main_mtr.dat","w")) == 0){
            printf("Unable to open 'main_mtr.dat'\n");
            exit(-1);
        }

        fprintf(outfp,"int main_mtr[141] = {\n");
        for (rpm = -700.0; rpm <= 700.0; rpm += 10.0){
            calc_rpm(rpm);
        }
        fprintf(outfp,"};\n");
        fclose(outfp);
} /* CALC_MN.C

/*
 *     S.DAT - servo lookup table for AUV II
 *     Author - LT M.J. CLOUTIER
 */

int main_mtr[141] = {
        205,213,220,227,235,242,250,257,264,272,279,284,290,295,
```

```
        300,306,311,316,322,327,332,337,342,347,352,357,362,367,
        372,377,382,387,391,395,399,403,407,411,415,419,423,427,
        430,433,436,439,442,446,449,452,455,457,460,462,464,466,
        469,471,473,475,478,481,484,488,491,495,498,501,505,508,
        512,515,518,521,525,528,531,534,538,541,544,547,549,552,
        554,557,559,561,564,566,569,572,576,579,583,586,590,593,
        597,600,604,609,614,619,624,629,634,639,644,649,654,659,
        664,670,675,680,685,690,696,701,706,712,717,723,729,734,
        740,745,751,757,762,771,779,788,796,805,813,822,830,839,
        847,
};
```

```
/*
 *    Program --- TEST_MN.C
 *    Purpose --- main motor test program for AUV II
 *    Author ---- LT M.J. CLOUTIER
 */

#include <math.h>
#include <stdio.h>
#include "m.dat"

/* address of ADA1 (could also use DAC2B) */
#define DAC_ADDR        0xFFF00000
#define DAC_LSB_OFFSET  0x2
/* PORT_MAIN is channel 1 of ADA1 */
#define PORT_MAIN  1

unsigned char *dac_a = DAC_ADDR;

/*********************************************************
 *  dac(s,ch) -- writes signal 's' to ada1 dac channel 'ch'
 *               (allowable channels 0-3)
 *********************************************************/
void dac(s,ch)
int s,ch;
{
      ch = ch << 2;                          /* offset for G-96 bus addressing */
      dac_a[ch] = s >> 2;                    /* write upper 8 bits to MSB */
      dac_a[ch + DAC_LSB_OFFSET] = s << 6;   /* write lower 2 bits B3,B2 */
   return;
} /* dac */

void main_rpm(motor,rpm)
int motor;
double rpm;
{
      rpm += 700.0;                          /* to normalize to 0-1400 RPM */
      dac(main_mtr[(int)(rpm/10.0)],motor);
}/* main_rpm */

main ()
{
      double rpm;                    /* RPM value to send to MAIN_RPM */
```

```
        rpm = 0.0;
        while ((rpm <= 700.0) && (rpm >= -700.0)){
            printf("Enter desired RPM value from -700 to +700 RPM (999 to quit)\n");
            scanf("%lf",&rpm);
            main_rpm(PORT_MAIN,rpm);
        }
        main_rpm(PORT_MAIN,0.0);
} /* TEST_MN.C */
```

APPENDIX E. GLOSSARY

Waypoint -

a quadruple (x,y,z,v) consisting of position at the intersection of grid points in a global rectangular three-dimensional grid coordinate system (x,y,z) and linear velocity (v) in the x direction of a vehicle centered coordinate system.

Posture -

a sextuple (x,y,z,θ,ϕ,v) consisting of position (x,y,z) in a three-dimensional rectangular coordinate system (not constrained to grid intersections), velocity as defined above, heading (θ) in the x-y plane, and pitch angle (ϕ) in the x-z plane.

Reference Posture -

Reference postures are generated by **STEP_SPIRAL.** They are postures which are located on the current cubic spiral somewhere between the last waypoint and the current waypoint. The velocity in a reference posture is taken directly from the current waypoint.

Desired Posture -

A Reference Posture which has had navigational compensation applied to position, angles and velocity based on actual vehicle position and velocity parameters. Desired Postures are generated by the **Tracking Controller** and are provided as input to the **Autopilot.**

Current Posture -

The current vehicle posture consists of current position and velocity as determined by **Navigation.**

APPENDIX F. OFF-LINE GUIDANCE SOURCE CODE

```
/*
 *    WAYPT.DAT -- the required input file for FIND_CS.C
 *                 These are the 27 possible combinations of grid points for AUV II
 */

0 0 1 1 0 2 -1 1
0 0 1 1 0 2 -1 2
0 0 1 1 0 2 -1 3

0 0 1 1 1 2 0 3
0 0 1 1 1 2 1 3
0 0 1 1 1 2 2 3

0 0 1 1 2 2 3 3
0 0 1 1 2 2 3 2
0 0 1 1 2 2 3 1

0 0 1 0 2 1 1 2
0 0 1 0 2 1 2 2
0 0 1 0 2 1 3 2

0 0 1 0 2 0 3 1
0 0 1 0 2 0 3 0
0 0 1 0 2 0 3 -1

0 0 1 0 2 -1 3 -2
0 0 1 0 2 -1 2 -2
0 0 1 0 2 -1 1 -2

0 0 1 -1 2 -2 3 -1
0 0 1 -1 2 -2 3 -2
0 0 1 -1 2 -2 3 -3

0 0 1 -1 1 -2 2 -3
0 0 1 -1 1 -2 1 -3
0 0 1 -1 1 -2 0 -3

0 0 1 -1 0 -2 -1 -3
```

```
0 0 1 -1 0 -2 -1 -2
0 0 1 -1 0 -2 -1 -1


/*********************************************************************
 *
 *    find_cs.c - This program is used to determine the appropriate
 *                cubic spiral path for the NPS AUV local path-planner.
 *
 *                It is a modification of the cubic spiral calculation
 *                program developed by Yutaka Kanayama for Yamabico-11.
 *
 *    LT M.J. CLOUTIER
 *    12/18/89
 *
 *    Inputs  - file consisting of sets of four waypoints
 *                (expressed as doubles) consisting of x and y position
 *
 *    Outputs - file consisting of one or more cubic spiral values
 *                for A and l
 *
 *********************************************************************/

#include <stdio.h>
#include <math.h>
#include "cst.h"

/* required for cot function (doesn't exist in TurboC) */
#define TURBOC 0

/*************************************************************
 *      MACROS
 *************************************************************/

#define SQR(x) ( (x) * (x) )
#define CUBE(x) ((x) * (x) * (x))
#define EUCL_DIST(x1,y1,x2,y2) (sqrt(((x1) - (x2)) * ((x1) - (x2)) + ((y1) - (y2)) *
((y1) - (y2))))
#define DIST(x1,y1,x2,y2) (fabs((x1) - (x2)) + fabs((y1) - (y2)))
#define r2d(r) ((r) * RAD)

/*************************************************************
 *      DEFINED CONSTANTS
```

```
 ************************************************************/

#define TRUE_    1
#define FALSE_   !TRUE_
#define SUCCESS  1
#define FAILURE  !SUCCESS

/************************************************************
 *      NUMERICAL CONSTANTS
 ************************************************************/

#define PI_          3.14159265358979323846
#define NPI         -3.14159265358979323846
#define DPI          6.28318530717958647692
#define HPI          1.57079632679489661923
#define RAD          57.29577951308232087684
#define ZERO_RAD     0.01
#define NAMESIZE     20

/************************************************************
 *      VARIABLES
 ************************************************************/

typedef struct posture {
      char   posture_id[NAMESIZE];
      double x;
      double y;
      double theta;
      double k;
}
POSTURE;

FILE *infp, *outfp;

/************************************************************
 *     zero_rad - TRUE if data is <= ZERO_RAD
 ************************************************************/
int zero_rad(data)
double data;
{
      return ((fabs(data) <= ZERO_RAD) ? TRUE_ : FALSE_);
} /* zero_rad */
```

```
/*************************************************************
 *    cot - returns the trigonometric cotangent
 *          (TurboC doesn't provide cot!!!)
 *************************************************************/
#ifdef TURBOC
double cot(arg)
double arg;
{
      double tmp;

      if (zero_rad(arg)) {
            fprintf(stderr,"cot:  bad argument %7.3f\n",arg);
            exit(-1);
      } else {
            return (1.0 / tan(arg));
      }
}
#endif
/* cot */

/*************************************************************
 *    create_posture - assigns x,y,theta to posture
 *************************************************************/
POSTURE *create_posture(x,y,theta,p)
double x, y;
double theta;
POSTURE *p;
{
      p->x = x;
      p->y = y;
      p->theta = theta;
      return(p);
} /* create_posture */

/*************************************************************
 *    pnorm - positive normalization (0 < a < 2PI)
 *************************************************************/
double pnorm(a)
double a;
{
      while (a >= DPI)
          a -= DPI;
       while (a < 0)
```

```
        a += DPI;
    return(a);
} /* pnorm */

/*************************************************************
 *    norm - normalizes angle between -PI and PI
 *************************************************************/
double norm(a)
double a;
{
    while (a > PI_)
        a -= DPI;
    while (a <= NPI)
        a += DPI;
    return(a);
} /* norm */

/*********************************************************************
 *
 *    LOOKUP_CURVE - returns normalized distance for value looked
 *                   up in cst.h
 *
 *                     \
 *       cst[a1] |------- \
 *               |        | \
 *          dist |--------|---\
 *               |        |   | \
 *       cst[a2] |--------|---|---\
 *               |________|___|___|_\____
 *                     a1  tmp  a2
 *
 *********************************************************************/

double lookup_curve(alfa)
double alfa;
{
    int alfa1, alfa2;
    double dist, tmp;

    tmp = r2d(alfa);        /* because cst.h is in degrees */
    alfa1 = (int) tmp;      /* get cst value below tmp */
    alfa2 = alfa1 + 1;      /* get cst value above tmp */
    tmp = tmp - alfa1;      /* set tmp to delta btwn alfa1 and tmp */
```

```
    if (alfa1 <= 281) {
        /* interpolate dist based on proximity of alfa to integer values */
        dist = cst[alfa2] * tmp + cst[alfa1] * (1 - tmp);
    } else { /* zero crossing undefined -> set to 1000 */
        dist = 1000;
    }
    return (dist);

} /* lookup_curve */

/*****************************************************************
 *
 *    COSTF - used to calculate best split point when two CS are
 *            required. Returns the local cost of an ALFA,DIST pair
 *            where cost is ratio of angle squared to length cubed
 *            (from Smooth Path Planning Paper)
 *
 *            Minimum cost occurs if ANGLE is ZERO. Minimum cost
 *            for a given angle occurs when length is large (small
 *            curvature)
 *
 *                len        1              dist
 *               ------  =  ------   => len ≈ -----
 *                dist      D(alfa)          D(alfa)
 *
 *****************************************************************/

double costf(alfa,dist)
double alfa, dist;
{
    return(SQR(alfa) * CUBE(lookup_curve(fabs(alfa))/dist));
} /* costf */

/*****************************************************************
 *
 *    COST - returns the cost of a given trajectory, where cost
 *           is defined to be sum of the costs of two separate
 *           DIST,ALPHA pairs.
 *
 *           xc,yc are center of circle which is locus of mid-points
 *           symmetric to both (x1,y1) and (x2,y2).
 *           r is radius of circle. g is angle to r
 *
```

```
 *
 *                        x (xc,yc)
 *            (x1,y1)       |          /|\
 *             x ->         |           x (x2,y2)
 *               \          |         /
 *                 \        | r     /
 *                   \      |     /
 *               dist1 \    |   / dist2
 *                       \  | /
 *                        \ /
 *
 ************************************************************************/

double cost(x1,y1,theta1,x2,y2,theta2,xc,yc,r,g)
double x1,y1,theta1,x2,y2,theta2,xc,yc,r,g;
{
      double x,y,d,dist1,dist2,alpha1,alpha2;

      /* find center of circle */
      x = xc + r * cos(g);
      y = yc + r * sin(g);
      dist1 = EUCL_DIST(x1,y1,x,y);
      dist2 = EUCL_DIST(x2,y2,x,y);

      /* 2.0 * forces symmetry (SPP Paper) */
      alpha1 = 2.0 * norm(atan2(y - y1, x - x1) - theta1);
      alpha2 = 2.0 * norm(atan2(y2 - y, x2 - x) - theta2);

      /* Total cost is cost of both CS */
      return(costf(alpha1,dist1) + costf(alpha2,dist2));
} /* cost */

/************************************************************************
 *
 *    SPLIT - splits a set of postures that require two cubic spirals
 *            into two separate postures with the proper mid-point
 *
 ************************************************************************/

POSTURE *split(p_, q_, Q)
POSTURE *p_, *q_, *Q;
{
      double co, xc, yc, r, xm, ym, theta_mid, g1, g2, g, w;
```

```
double x1, y1, theta1, x2, y2, theta2, alpha, alpha1, beta1, eta1, eta2, w1;
double costg, costg1,costg2;

x1 = p_->x;
y1 = p_->y;
theta1 = p_->theta;
x2 = q_->x;
y2 = q_->y;
theta2 = q_->theta;

/* determine if symmetric/parallel */
alpha = norm(theta2 - theta1);
alpha1 = fabs(alpha);

/* if the postures are parallel then split at midpoint of line between */
if (alpha1 < ZERO_RAD) {
    xm = (x1 + x2) / 2.0;
    ym = (y1 + y2) / 2.0;

} else {  /* find the best midpoint based on minimum cost */
    /* first find the center of locus of valid midpoints */
    co = cot(alpha / 2.0);
    xc = (x1 + x2 + co * (y1 - y2)) / 2.0;
    yc = (y1 + y2 + co * (x2 - x1)) / 2.0;
    r = EUCL_DIST(x1,y1,xc,yc);

    /*  now pick proper values for quadrant */
    if (alpha > 0.0) {
        g1 = atan2(y1 - yc, x1 - xc);
        g2 = atan2(y2 - yc, x2 - xc);
        eta1 = theta1 - HPI;
        eta2 = theta2 - HPI;
    } else {
        g1 = atan2(y2 - yc, x2 - xc);
        g2 = atan2(y1 - yc, x1 - xc);
        eta1 = theta2 + HPI;
        eta2 = theta1 + HPI;
    }

    /* g is direction to r from (xc,yc), w is increment for search */
    g = (g1 + g2) / 2.0;
    w = alpha1 / 2.0;
```

```
        /* pick proper chunk of curve to explore (to minimize search time */
        if (((w1 = pnorm(eta1 - g1)) < alpha1) && (2.0 * w1 < alpha1)) {
            g = eta1;
            w = w1;
        }
        if (((w1 = pnorm(g2 - eta2)) < alpha1) && (2.0 * w1 < alpha1)) {
            g = eta2;
            w = w1;
        }

        /* iterate g until get minimum cost */
        costg= cost(x1,y1,theta1,x2,y2,theta2,xc,yc,r,g);
        while (w > ZERO_RAD) {
            w = w / 2.0;
            costg1= cost(x1,y1,theta1,x2,y2,theta2,xc,yc,r,g+w);
            if(costg1 < costg) {
                g = g + w;
                costg = costg1;
            } else {
                costg2 = cost(x1,y1,theta1,x2,y2,theta2,xc,yc,r,g-w);
                if (costg2 < costg) {
                    g = g - w;
                    costg = costg2;
                }
            }
        };

        /* now calculate midpoint for split */
        xm = xc + r * cos(g);
        ym = yc + r * sin(g);
    }

    /* create new posture for midpoint of split */
    beta1= atan2(ym - y1, xm - x1);
    theta_mid = beta1 + norm(beta1 - theta1);
    create_posture(xm,ym,theta_mid,Q);
    return(Q);
} /* split */


/*****************************************************************
 *    SOLVE1 - gives simple turn solution using cubic spiral
 *****************************************************************/
```

```
solve1(p, q)
POSTURE *p, *q;
{
        double alpha, len;

        /* these calculations are same as above (COST) from SPP Paper */
        alpha = 2 * norm(atan2(q->y - p->y, q->x - p->x) - p->theta);
        len = EUCL_DIST(q->x, q->y, p->x, p->y) / lookup_curve(fabs(alpha));

        /* write into instruction buffer  */
        fprintf(outfp,"%16.12f,%16.12f",len,6*alpha/(CUBE(len)));
} /* solve1 */

/*********************************************************************
 *      SOLVE - returns a single posture (or posture pair) based on input
 *              pairs of (x,y,theta)
 *********************************************************************/

int solve(pp, qq)
POSTURE *pp, *qq;
{
        POSTURE p,q,midpst;
        double beta;

        p = *pp;
        q = *qq;

        /* check if valid input */
        if ((zero_rad(p.y - q.y)) && (zero_rad(p.x - q.x))) {
                fprintf(stderr,"Bad waypts (%d,%d) (%d,%d)\n",p.x,p.y, q.x,q.y);
                exit(-1);
        }

        /* find angle between postures */
        beta = atan2((q.y - p.y), (q.x - p.x));

        /* if symmetric use one CS */
        fprintf(outfp,"\t{");
        if (fabs(norm(q.theta - beta) - norm(beta - p.theta)) < ZERO_RAD) {
                solve1(pp, qq);
                fprintf(outfp,",%16.12f,%16.12f",0.0,0.0);
        } else { /* split and use two cubic spirals*/
                split(pp, qq, &midpst);
```

```
            solve1(pp, &midpst);
            fprintf(outfp,",");
            solve1(&midpst, qq);
        }
        fprintf(outfp,"},\n");
        return (SUCCESS);
} /* solve */


main(argc,argv)
int  argc;
char *argv[];
{
        int scan_result;
        double x1,y1,x2,y2,x3,y3,x4,y4;
        double theta1, theta2;
        POSTURE p,q;

        /* open input and output files */
        if ((infp = fopen(argv[1],"r")) == 0) {
            printf("Unable to open %s for input\n",argv[1]);
            exit(-1);
        }
        if ((outfp = fopen(argv[2],"w")) == 0) {
            printf("Unable to open %s for output\n",argv[2]);
            exit(-1);
        }

        /* get each set of waypts */
        while ((scan_result = fscanf(infp,"%lf %lf %lf %lf %lf %lf %lf %lf",
            &x1,&y1,&x2,&y2,&x3,&y3,&x4,&y4)) != EOF) {

            /* Determine the angles */
            theta1 = atan2((y3 - y1),(x3 - x1));
            theta2 = atan2((y4 - y2),(x4 - x2));

            /* create the postures */
            p.x = x2;
            p.y = y2;
            p.theta = theta1;
            q.x = x3;
            q.y = y3;
            q.theta = theta2;
```

```
            /* and calculate 'A' and 'l' */
            solve(&p,&q);
        }
    } /* main */

    /* find_cs.c */
```

APPENDIX G. REAL-TIME GUIDANCE SOURCE CODE

```
/*************************************************************
 *
 *      lpp.h - Header file for NPS AUV Local Path-Planner
 *              (adapted from mml.h for Yamabico-11 by
 *               Yutaka Kanayama)
 *
 *      LT M.J. CLOUTIER
 *      12/19/89
 *
 *      Rev1 -  1/24/90
 *
 *************************************************************/

/*************************************************************
 *      MACROS
 *************************************************************/

#define SQR(x) ( (x) * (x) )
#define CUBE(x) ((x) * (x) * (x))
#define EUCL_DIST(x1,y1,x2,y2) (sqrt(((x1) - (x2)) * ((x1) - (x2)) + ((y1) - (y2)) *
((y1) - (y2))))
#define DIST(x1,y1,x2,y2) (fabs((x1) - (x2)) + fabs((y1) - (y2)))
#define SQRT(xc) (sqrt(1.0 + (xc)*(xc)))
#define SIGN(x)  (((x) < 0.0) ? 0x01 : ((x) > 0.0) ? 0x02 : 0x00)
#define r2d(r) ((r) * RAD)
#define d2r(d) ((d) / RAD)

/*************************************************************
 *      DEFINED CONSTANTS
 *************************************************************/

#define TRUE_    1
#define FALSE_   !TRUE_
#define ON       1
#define OFF      !ON
#define YES      1
#define NO       !YES
#define SUCCESS  1
```

```
#define FAILURE  !SUCCESS

/* THESE MAY BE CHANGED AS REQUIRED */
#define GRID_SZ 180          /* 70in ~= 180cm */
#define MAX_STEPS 200
#define CLOSE_ENOUGH 0.5
#define INTRVL 0.0001
#define RPM_2_VEL  0.04921   /* in/sec*sec/min*min/rev*cm/in (350rpm=4fps)*/
#define K_X   10
#define K_Y   0.0064
#define K_HDG 0.16
#define MIN_X_ERROR 1

#define CSTA     361        /* size of cubic spiral table */
#define BUFSIZE  1024       /* buffer size for getstr */
#define MAXLINE  40         /* max string length */
#define NAMESIZE 20         /* name size for posture */

/*************************************************************
 *      NUMERICAL CONSTANTS
 *************************************************************/

#define PI_         3.14159265358979323846
#define DPI         6.28318530717958647692          /* PI*2   */
#define RAD         57.29577951308232087684          /* 180/PI */
#define HPI         1.57079632679489661923          /* PI/2   */
#define QPI         0.78539816339744830962          /* PI/4 (was PI4) */
#define ZERO_RAD    0.01
#define NO_SPIRAL 0
#define SINGLE    1
#define DOUBLE1   2
#define DOUBLE2   3

#define N       0x01
#define Z_      0x00
#define P       0x02

#define NN      0x05
#define NZ      0x04
#define NP      0x06
#define ZN      0x01
#define ZZ      0x00
#define ZP      0x02
```

```
#define PN      0x09
#define PZ      0x08
#define PP      0x0a

#define NEG     -1
#define ZERO    0
#define POS     1

#define LFT    7
#define CENTER  0
#define RGHT   -7

#define L       0x01
#define C       0x00
#define R       0x02

#define LLL     0x15
#define LLC     0x14
#define LLR     0x16
#define LCL     0x11
#define LCC     0x10
#define LCR     0x12
#define LRL     0x19
#define LRC     0x18
#define LRR     0x1a
#define CLL     0x05
#define CLC     0x04
#define CLR     0x06
#define CCL     0x01
#define CCC     0x00
#define CCR     0x02
#define CRL     0x09
#define CRC     0x08
#define CRR     0x0a
#define RLL     0x25
#define RLC     0x24
#define RLR     0x26
#define RCL     0x21
#define RCC     0x20
#define RCR     0x22
#define RRL     0x29
#define RRC     0x28
#define RRR     0x2a
```

```
/*  lpp.h */


/*****************************************************************
 *
 *     curves.def - xy-plane curve descriptors for 'guidance.c'
 *
 *****************************************************************/
struct DESCRIPTOR{
      double l1,A1,l2,A2;
};

struct DESCRIPTOR desc[43] = {
   {  1.000000000000,   0.000000000000},
   {  0.426861119501, -20.677652279190,   0.600965473691,  20.227029553329},
   {  0.426861119501,  20.677652279190,   0.600965473691, -20.227029553329},
   {  0.000000000000,   0.000000000000},
   {  1.450450440178,   1.265295220167},
   {  0.475286674856,  -6.072725411590,   1.082989700434,   5.743097253612},
   {  0.832934089063,   5.272056359189,   0.600005018599,  -5.166871437198},
   {  0.000000000000,   0.000000000000},
   {  1.450450440178,  -1.265295220167},
   {  0.832934089063,  -5.272056359189,   0.600005018599,   5.166871437198},
   {  0.475286674856,   6.072725411590,   1.082989700434,  -5.743097253612},
   {  0.000000000000,   0.000000000000},
   {  0.000000000000,   0.000000000000},
   {  0.000000000000,   0.000000000000},
   {  0.000000000000,   0.000000000000},
   {  0.000000000000,   0.000000000000},
   {  0.600965473691,  20.227029553329,   0.426861119501, -20.677652279190},
   {  1.054318417319,   4.747378858590},
   {  0.527159208659,  37.979030868724,   0.527159208659, -37.979030868724},
   {  0.000000000000,   0.000000000000},
   {  1.082989700434,   5.743097253612,   0.475286674856,  -6.072725411590},
   {  1.652500089655,   2.088558538488},
   {  0.902131678928,  10.078999384402,   0.628250853818, -10.838140077661},
   {  0.000000000000,   0.000000000000},
   {  0.600005018599,   5.166871437198,   0.832934089063,  -5.272056359189},
   {  1.414213562373,   0.000000000000},
   {  0.628250853818,  10.838140077661,   0.902131678928, -10.078999384402},
   {  0.000000000000,   0.000000000000},
   {  0.000000000000,   0.000000000000},
```

```
    {  0.000000000000,   0.000000000000},
    {  0.000000000000,   0.000000000000},
    {  0.000000000000,   0.000000000000},
    {  0.600965473691, -20.227029553329,   0.426861119501,  20.677652279190},
    {  0.527159208659, -37.979030868724,   0.527159208659,  37.979030868724},
    {  1.054318417319,  -4.747378858590},
    {  0.000000000000,   0.000000000000},
    {  0.600005018599,  -5.166871437198,   0.832934089063,   5.272056359189},
    {  0.628250853818, -10.838140077661,   0.902131678928,  10.078999384402},
    {  1.414213562373,  -0.000000000000},
    {  0.000000000000,   0.000000000000},
    {  1.082989700434,  -5.743097253612,   0.475286674856,   6.072725411590},
    {  0.902131678928, -10.078999384402,   0.628250853818,  10.838140077661},
    {  1.652500089655,  -2.088558538488}
};


/*****************************************************************************
 *
 *     guidance.c -  This program is the local path-planner for the NPS
 *                   AUV.  It reads in waypoints from an input file,
 *                   determines the appropriate curve or curve pair for
 *                   the given waypoints and then calls the stepper.  It
 *                   writes to two separate files as indicated below
 *
 *     LT M.J. CLOUTIER
 *
 *     Inputs  - file (argv[1]) of waypoints consisting of x,y position
 *
 *     Outputs - file (argv[2]) of input descriptors for stepper
 *             - file (argv[3]) of stepper output postures
 *
 *     Rev4    -  1/11/90 - changed get_init_data to return (x0,y0,card_hdg)
 *     Rev3    -  1/10/90 - changed dir_chg to operate on sign of norm(hdg)
 *                        - changed get_descriptor to tbl lookup (vs switch)
 *                        - default cardinal_hdg is now determined from first
 *                          two way-pts
 *                        - eliminated atan2 using tbl lookup
 *                        - shifted output coord system [abs(HPI-theta)]
 *     Rev2    -  1/09/90 - incorporated GRID_SZ
 *     Rev1    -  1/07/90 - initialized ref_theta, init_hdg
 *     Orig    - 12/21/89
 *
```

```
 *
 ************************************************************************/

#include <stdio.h>
#include <math.h>
#include "lpp.h"
#include "curves.def"

double act_x, act_y, act_z, act_v, act_hdg;
#define act_omega 0.0

extern struct DESCRIPTOR;                    /* (from 'curves.def') */

/* exported GLOBALS  */
double des_x, des_y, des_z, des_v, des_hdg;  /* values to AUTOPILOT */

/* globals */
int x_last,y_last,z_last,v_last;/* holds next goal values from MP */
int x_next,y_next,z_next;       /* current goal position from MP */
int v_next;                     /* next desired velocity from MP */
int v_curr;                     /* current velocity */
int triple;                     /* the current turn combo (eg. LLL) */
double cardinal_hdg;            /* vehicle cardinal heading (N,S,E,W) */
struct DESCRIPTOR cd;           /* descriptor for current spiral */
double dist_trvl;               /* distance traveled on current spiral */
double st_x, st_y, st_hdg;      /* values from stepper */
FILE *infp, *outfp;             /* input path, output posture files */
int eof_infp;                   /* eof pointer for input */
double x_start, y_start, z_start; /* initial position */
int curve_type;                 /* (NO_SPIRAL,SINGLE,DOUBLE1,DOUBLE2) */

/*************************************************************
 *   MISCELLANEOUS MATH FUNCTIONS
 *************************************************************/

/*************************************************************
 *
 *      ATAN2 - returns table lookup of angle to replace
 *              atan2
 *
 *************************************************************/

/* TABLE LOOKUP VALUES */
```

```
double xy_ang[5][5] = {
   -2.356194490192344840,
   -2.034443935795702710,
   -1.570796326794896560,
   -1.107148717794090410,
   -0.785398163397448279,
   -2.677945044588986970,
   -2.356194490192344840,
   -1.570796326794896560,
   -0.785398163397448279,
   -0.463647609000806094,
    3.141592653589793120,
    3.141592653589793120,
    0.000000000000000000,
    0.000000000000000000,
    0.000000000000000000,
    2.677945044588986970,
    2.356194490192344840,
    1.570796326794896560,
    0.785398163397448279,
    0.463647609000806094,
    2.356194490192344840,
    2.034443935795702710,
    1.570796326794896560,
    1.107148717794090410,
    0.7853981633974482791};

double ATAN2(y2,y1,x2,x1)
int y2,y1,x2,x1;
{
   return xy_ang[(y2-y1)/GRID_SZ+2][(x2-x1)/GRID_SZ+2];
} /* ATAN2 */

/*********************************************************
 *
 *      norm - normalizes angle between -PI and PI
 *
 *********************************************************/
double norm(a)
double a;
{
   while ((a > PI_) || (a <= -PI_))
   {
```

```
        if (a > PI_)
            a = a - DPI;
        else
            a = a + DPI;
    }
    return (a);
} /* norm */

/*************************************************************
 *
 *      dir_chg - returns the relative direction chg (L,C or R)
 *                based on input waypoints. dir_chg uses the
 *                global "cardinal_hdg" to determine rel chg
 *
 *************************************************************/
int dir_chg(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
    double heading;
    int rel_dir;

    heading = ATAN2(y2,y1,x2,x1);
    heading -= cardinal_hdg;
    rel_dir = (int)(10.0*norm(heading));
    switch (rel_dir){
        case LFT    : cardinal_hdg += HPI;
                      return L;
        case CENTER  : return C;
        case RGHT    : cardinal_hdg -= HPI;
                      return R;
        default : fprintf(stderr,"dir_chg:  Bad result in dir_chg\n");
                  exit(-1);
    }
} /* dir_chg */

/*************************************************************
 *
 *   process_next_waypt - reads next waypt from infp and returns
 *                        the next curve descriptor
 *
 *************************************************************/
void process_next_waypt()
{
```

```
    int mp_x, mp_y, mp_z, mp_v;     /* values from MISSION PLANNER */

    if ((eof_infp = fscanf(infp,"%d %d %d %d",&mp_x,&mp_y,&mp_z,&mp_v)) !=
EOF){
        triple = (triple & 0x0f) << 2 | dir_chg(x_last,y_last,mp_x,mp_y);
        cd = desc[triple];
        v_curr = v_next; /* this is target velocity */

        x_next = x_last; /* this is current goal */
        y_next = y_last;
        z_next = z_last;
        v_next = v_last;

        x_last = mp_x;   /* this is next goal */
        y_last = mp_y;
        z_last = mp_z;
        v_last = mp_v;

        /* for now the desired depth is the same as reference */
        des_z = (double)z_next;
    }
} /* process_next_waypt */

/*************************************************************
 *
 *      next_pos - calculates next reference values for
 *                 x, y and theta
 *
 *************************************************************/
void next_pos(A,l)
double A,l;
{
    double theta, del_theta, step, t_step;
    double kappa;

/* $$$$$$$$ DEBUGGING VALUES $$$$$$$$ */
act_x = st_x;
act_y = st_y;
act_hdg = st_hdg;
/* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ */

    step = des_v * RPM_2_VEL * INTRVL;
    dist_trvl += step;
```

```
    kappa = A * dist_trvl * (1 - dist_trvl);
    del_theta = kappa * step;
    st_hdg += del_theta;
    theta = st_hdg - del_theta/2.0;
    if (fabs(del_theta) == 0.0){
        t_step = step;
    } else {
        t_step = step * (sin(del_theta / 2.0)/(del_theta / 2.0));
    }
    st_x += cos(theta) * t_step;
    st_y += sin(theta) * t_step;
} /* next_pos */

/*************************************************************
 *
 *      print_stepper_out - prints x,y and theta to step_outfp
 *
 *************************************************************/
void print_stepper_out()
{
    fprintf(outfp,"%d %d %8.4f\n",
        st_x, st_y, norm(fabs(HPI - st_hdg)));
} /* print_stepper_out */

/*************************************************************
 *
 *      print_desired_out - prints x,y and theta to step_outfp
 *
 *************************************************************/
void print_desired_out()
{
/*
    fprintf(outfp,"%8.4f %8.4f %8.4f %8.4f %8.4f\n",
        des_x + x_start, des_y + y_start, des_z + z_start,
        des_v ,norm(fabs(HPI - des_hdg)));
*/
    fprintf(outfp,"%8.4f %8.4f\n",
        des_x + x_start, des_y + y_start);
} /* print_desired_out */

/*************************************************************
 *
 *      step_spiral - breaks spiral into discrete postures,
```

```
 *                prints to output file with break between
 *                curves
 *
 ************************************************************/
void step_spiral()
{
   int step_ct;

   step_ct = 0;
   while(step_ct < MAX_STEPS){
      switch(curve_type){
         case NO_SPIRAL: dist_trvl = 0.0;
                         /* get spiral type */
                         if (cd.l2 == 0.0){
                            curve_type = SINGLE;
                         } else {
                            curve_type = DOUBLE1;
                         }
                         /* step to next_pos */
                         for (;((dist_trvl <= cd.l1) && (step_ct < MAX_STEPS));
step_ct++){
                            next_pos(cd.A1,cd.l1);
                         }
                         break;
         case SINGLE:
         case DOUBLE1: for(;((dist_trvl <= cd.l1) && (step_ct < MAX_STEPS));
step_ct++){
                         next_pos(cd.A1,cd.l1);
                       }
                       /* finished this spiral, reset dist_trvled, do next */
                       if (dist_trvl >= cd.l1){
                          dist_trvl = 0.0;
                          /* if second part exists => start along it */
                          if (cd.l2 != 0.0){
                             curve_type = DOUBLE2;
                             for (;((dist_trvl <= cd.l2)&&(step_ct < MAX_STEPS));
step_ct++){
                                next_pos(cd.A2,cd.l2);
                             }
                          } else { /* go to next way_pt */
                             curve_type = NO_SPIRAL;
                          }
                       }
```

```
                    break;
        case DOUBLE2:       for (;((dist_trvl <= cd.l2) && (step_ct <
MAX_STEPS)); step_ct++){
                        next_pos(cd.A2,cd.l2);
                    }
                    if (dist_trvl >= cd.l2){
                        dist_trvl = 0.0;
                        curve_type = NO_SPIRAL;
                    }
                    break;
        default:      fprintf(stderr,"step_spiral:  Bad input value\n");
      }
      if (curve_type == NO_SPIRAL){
        process_next_waypt();
      }
   }
printf(" lpp (x,y) (%4.3f,%4.3f) MP (x,y) (%d,%d)\n",
   st_x*(double)GRID_SZ+x_start,st_y*(double)GRID_SZ+y_start,x_next,y_next);
   print_desired_out();
} /* step_spiral */

/*************************************************************
 *
 *     calc_des_val() - finds error values and desired values
 *                      to feed to AUTOPILOT
 *
 *************************************************************/
void calc_des_val()
{
   double err_x, err_y, err_hdg;

   err_x = (((st_x - act_x)*cos(act_hdg)) + ((st_y - act_y)*sin(act_hdg)))/2.57;
   err_y = (((act_x - st_x)*sin(act_hdg)) + ((st_y - act_y)*cos(act_hdg)))/2.57;
   err_hdg = norm(st_hdg - act_hdg);
   des_x = (st_x + err_x) * (double)GRID_SZ;
   des_y = (st_y + err_y) * (double)GRID_SZ;
   des_v = (K_X * err_x) + ((double)v_curr * cos(err_hdg));
   des_hdg += INTRVL*(act_omega + (v_curr * (K_Y * err_y + K_HDG *
err_hdg)));
   /* if not too far behind, then get next st_ values else use old ones */
   if (err_x <= MIN_X_ERROR){
      step_spiral();
   }
```

```
} /* calc_des_val */

/*************************************************************
 *
 *      open_files() - open input & output files
 *
 *************************************************************/
void open_files()
{
    if ((infp = fopen("path.in","r")) == 0) {
        printf("Unable to open 'path.in'\n");
        exit(-1);
    }
    if ((outfp = fopen("path.out","w")) == 0) {
        printf("Unable to open 'path.out'\n");
        exit(-1);
    }
} /* open_files */

/*************************************************************
 *
 *      get_init_values - get initial lpp values
 *
 *************************************************************/
void get_init_values()
{
    int dx1,dy1,dx2,dy2;
    int dir1,dir2,dir3;
    int x_curr, y_curr, z_curr;
    int x_past, y_past, z_past;

    /* initialize variables */
    x_past = y_past = 0;
    x_curr = y_curr = z_curr = v_curr = 0;
    x_next = y_next = z_next = v_next = 0;
    x_last = y_last = z_last = v_last = 0;

    /* this is the first time through, so read four waypoints */
    if ((eof_infp = fscanf(infp,"%d %d %d %d  %d %d %d %d  %d %d %d %d",
        &x_curr,&y_curr,&z_curr,&v_curr,
        &x_next,&y_next,&z_next,&v_next,
        &x_last,&y_last,&z_last,&v_last)) != EOF) {
```

```
/* get initial data */
dx1 = SIGN(x_next - x_curr);
dy1 = SIGN(y_next - y_curr);
dx2 = SIGN(x_last - x_next);
dy2 = SIGN(y_last - y_next);
x_start = x_curr;
y_start = y_curr;
z_start = z_curr;

/* this switch determines a valid position PRIOR to start */
switch((dx1 << 2) | dy1){
   case PZ :  cardinal_hdg = ZERO;
              x_past = x_curr - GRID_SZ;
              y_past = y_curr;
              break;
   case PP :  switch ((dx2 << 2) | dy2){
                 case PN : cardinal_hdg = HPI;
                           x_past = x_curr;
                           y_past = y_curr - GRID_SZ;
                           break;
                 default : cardinal_hdg = ZERO;
                           x_past = x_curr - GRID_SZ;
                           y_past = y_curr;
              }
              break;
   case PN :  switch ((dx2 << 2) | dy2){
                 case PP : cardinal_hdg = -HPI;
                           x_past = x_curr;
                           y_past = y_curr + GRID_SZ;
                           break;
                 default : cardinal_hdg = ZERO;
                           x_past = x_curr - GRID_SZ;
                           y_past = y_curr;
              }
              break;
   case ZP :  cardinal_hdg = HPI;
              x_past = x_curr;
              y_past = y_curr - GRID_SZ;
              break;
   case ZN :  cardinal_hdg = -HPI;
              x_past = x_curr;
              y_past = y_curr + GRID_SZ;
              break;
```

```
        case NZ :   cardinal_hdg = PI_;
                    x_past = x_curr + GRID_SZ;
                    y_past = y_curr;
                    break;
        case NP :   switch ((dx2 << 2) | dy2){
                        case NN : cardinal_hdg = HPI;
                                  x_past = x_curr;
                                  y_past = y_curr - GRID_SZ;
                                  break;
                        default : cardinal_hdg = PI_;
                                  x_past = x_curr + GRID_SZ;
                                  y_past = y_curr;
                    }
                    break;
        case NN :   switch ((dx2 << 2) | dy2){
                        case NP : cardinal_hdg = -HPI;
                                  x_past = x_curr;
                                  y_past = y_curr + GRID_SZ;
                                  break;
                        default : cardinal_hdg = PI_;
                                  x_past = x_curr + GRID_SZ;
                                  y_past = y_curr;
                    }
                    break;
        default : fprintf(stderr,"get_init_hdg: Bad input\n");
                  exit(-1);
    }

    /* now get triple */
    dir1 = dir_chg(x_past,y_past,x_curr,y_curr);
    dir2 = dir_chg(x_curr,y_curr,x_next,y_next);
    dir3 = dir_chg(x_next,y_next,x_last,y_last);
    triple = (dir1 << 4) | (dir2 <<2) | dir3;

    /* and 'current' vehicle heading */
    st_hdg = ATAN2(y_next,y_past,x_next,x_past);

    /* for now the desired depth is MP depth */
    des_hdg = st_hdg;
    des_z = (double) z_next;
    des_v = (double) v_curr;
} else {
    fprintf(stderr,"lpp_step main: not enough waypoints for path\n");
```

```
            exit(-1);
        }
        /* read curve information */
        cd = desc[triple];
        /* and get first desired values */
        step_spiral();
    } /* get_init_values */

    main()
    {
        int setparm = TRUE_;

        while (eof_infp != EOF){
            if (setparm){
                open_files();
                get_init_values();
                setparm = FALSE_;
            }
            calc_des_val();
        }
        close(infp);
        close(outfp);
    }

    /* guidance.c */
```

APPENDIX H. THREE DIMENSIONAL SOURCE CODE

```
/*    D10_70.DAT  */

0.0  0.0        1.0  0.0   2.0  0.03.0  0.0
0.0  0.0        1.0  0.0   2.0  0.03.0  0.142857142857
0.0  0.0        1.0  0.0   2.0  0.03.0 -0.142857142857

0.0  0.0        1.0  0.0   2.0  0.1428571428573.0  0.142857142857
0.0  0.0        1.0  0.0   2.0  0.1428571428573.0  0.285714285714
0.0  0.0        1.0  0.0   2.0  0.1428571428573.0  0.0

0.0  0.0        1.0  0.0   2.0 -0.1428571428573.0 -0.142857142857
0.0  0.0        1.0  0.0   2.0 -0.1428571428573.0  0.0
0.0  0.0        1.0  0.0   2.0 -0.1428571428573.0 -0.285714285714


0.0  0.0        1.0  0.1428571428572.0  0.1428571428573.0  0.142857142857
0.0  0.0        1.0  0.1428571428572.0  0.1428571428573.0  0.285714285714
0.0  0.0        1.0  0.1428571428572.0  0.1428571428573.0  0.0

0.0  0.0        1.0  0.1428571428572.0  0.2857142857143.0  0.285714285714
0.0  0.0        1.0  0.1428571428572.0  0.2857142857143.0  0.428571428571
0.0  0.0        1.0  0.1428571428572.0  0.2857142857143.0  0.142857142857

0.0  0.0        1.0  0.1428571428572.0  0.03.0  0.0
0.0  0.0        1.0  0.1428571428572.0  0.03.0  0.142857142857
0.0  0.0        1.0  0.1428571428572.0  0.03.0 -0.142857142857


0.0  0.0        1.0 -0.1428571428572.0 -0.1428571428573.0 -0.142857142857
0.0  0.0        1.0 -0.1428571428572.0 -0.1428571428573.0  0.0
0.0  0.0        1.0 -0.1428571428572.0 -0.1428571428573.0 -0.285714285714

0.0  0.0        1.0 -0.1428571428572.0  0.03.0  0.0
0.0  0.0        1.0 -0.1428571428572.0  0.03.0  0.142857142857
0.0  0.0        1.0 -0.1428571428572.0  0.03.0 -0.142857142857

0.0  0.0        1.0 -0.1428571428572.0 -0.2857142857143.0 -0.285714285714
0.0  0.0        1.0 -0.1428571428572.0 -0.2857142857143.0 -0.142857142857
```

```
0.0  0.0          1.0  -0.1428571428572.0 -0.2857142857143.0 -0.428571428571

/*************************************************************************
 *
 *   d_10-70.def -- curve descriptors based on 10/70 grid
 *
 *************************************************************************/

struct DESCRIPTOR d_10[43] = {
      {  1.000000000000,  0.000000000000,  0.000000000000,  0.000000000000},
      {  0.375115235242, -5.066081287242,  0.625593247974,  2.839641101518},
      {  0.375115235242,  5.066081287242,  0.625593247974, -2.839641101518},
      {  0.505688790039,  6.550461901757,  0.505688790039, -6.550461901757},
      {  0.631932817268,  2.727297694553,  0.378921326300, -4.865469783722},
      {  0.506118022287,  8.183874149810,  0.507051887040,-11.420667239510},
      {  0.505688790039, -6.550461901757,  0.505688790039,  6.550461901757},
      {  0.506118022287, -8.183874149810,  0.507051887040, 11.420667239510},
      {  0.631932817268, -2.727297694553,  0.378921326300,  4.865469783722},
      {  0.625593247974, -2.839641101518,  0.375115235242,  5.066081287241},
      {  0.500619316543, -6.820142254254,  0.500619316543,  6.820142254254},
      {  1.001238633086, -0.852517781782,  0.000000000000,  0.000000000000},
      {  0.378921326300,  4.865469783722,  0.631932817268, -2.727297694553},
      {  1.010152544552,  0.000000000000,  0.000000000000,  0.000000000000},
      {  0.442414807789,  5.530421017324,  0.570239645211, -7.174182722803},
      {  0.507051887040,-11.420667239510,  0.506118022287,  8.183874149809},
      {  0.507556871107,-13.022683702248,  0.507556871107, 13.022683702248},
      {  0.570239645211, -7.174182722804,  0.442414807789,  5.530421017324},
      {  0.625593247974,  2.839641101518,  0.375115235242, -5.066081287241},
      {  1.001238633086,  0.852517781782,  0.000000000000,  0.000000000000},
      {  0.500619316543,  6.820142254254,  0.500619316543, -6.820142254254},
      {  0.507051887040, 11.420667239510,  0.506118022287, -8.183874149809},
      {  0.570239645211,  7.174182722804,  0.442414807789, -5.530421017324},
      {  0.507556871107, 13.022683702248,  0.507556871107,-13.022683702248},
      {  0.378921326300, -4.865469783722,  0.631932817268,  2.727297694553},
      {  0.442414807789, -5.530421017324,  0.570239645211,  7.174182722803},
      {  1.010152544552, -0.000000000000,  0.000000000000,  0.000000000000}
};

/*    D_7_70.dat */
0.0  0.0    1.0  0.0   2.0  0.03.0  0.0
0.0  0.0    1.0  0.0   2.0  0.03.0  0.1
0.0  0.0    1.0  0.0   2.0  0.03.0 -0.1
```

```
0.0  0.0    1.0  0.0   2.0  0.13.0  0.1
0.0  0.0    1.0  0.0   2.0  0.13.0  0.2
0.0  0.0    1.0  0.0   2.0  0.13.0  0.0

0.0  0.0    1.0  0.0   2.0 -0.13.0 -0.1
0.0  0.0    1.0  0.0   2.0 -0.13.0  0.0
0.0  0.0    1.0  0.0   2.0 -0.13.0 -0.2


0.0  0.0    1.0  0.1   2.0  0.13.0  0.1
0.0  0.0    1.0  0.1   2.0  0.13.0  0.2
0.0  0.0    1.0  0.1   2.0  0.13.0  0.0

0.0  0.0    1.0  0.1   2.0  0.23.0  0.2
0.0  0.0    1.0  0.1   2.0  0.23.0  0.3
0.0  0.0    1.0  0.1   2.0  0.23.0  0.1

0.0  0.0    1.0  0.1   2.0  0.03.0  0.0
0.0  0.0    1.0  0.1   2.0  0.03.0  0.1
0.0  0.0    1.0  0.1   2.0  0.03.0 -0.1


0.0  0.0    1.0 -0.1   2.0 -0.13.0 -0.1
0.0  0.0    1.0 -0.1   2.0 -0.13.0  0.0
0.0  0.0    1.0 -0.1   2.0 -0.13.0 -0.2

0.0  0.0    1.0 -0.1   2.0  0.03.0  0.0
0.0  0.0    1.0 -0.1   2.0  0.03.0  0.1
0.0  0.0    1.0 -0.1   2.0  0.03.0 -0.1

0.0  0.0    1.0 -0.1   2.0 -0.23.0 -0.2
0.0  0.0    1.0 -0.1   2.0 -0.23.0 -0.1
0.0  0.0    1.0 -0.1   2.0 -0.23.0 -0.3

/*****************************************************************************
 *
 *   d_7-70.def -- curve descriptors for depth based on 7/70 grid
 *
 *****************************************************************************/
struct DESCRIPTOR d_7[43] = {
     {  1.000000000000,  0.000000000000,  0.000000000000,  0.000000000000},
     {  0.375056870759, -3.550981208130,  0.625292414313,  1.992340756031},
     {  0.375056870759,  3.550981208130,  0.625292414313, -1.992340756031},
```

```
        { 0.502797441596,  4.692973208712,  0.502797441596, -4.692973208712},
        { 0.628408287342,  1.953101936801,  0.376926970544, -3.481012896273},
        { 0.503006506962,  5.864754182372,  0.503461901514, -8.197729062929},
        { 0.502797441596, -4.692973208712,  0.502797441596,  4.692973208712},
        { 0.503006506962, -5.864754182372,  0.503461901514,  8.197729062929},
        { 0.628408287342, -1.953101936801,  0.376926970544,  3.481012896273},
        { 0.625292414313, -1.992340756031,  0.375056870759,  3.550981208131},
        { 0.500305048597, -4.787238600005,  0.500305048597,  4.787238600005},
        { 1.000610097194, -0.598404825001,  0.000000000000,  0.000000000000},
        { 0.376926970544,  3.481012896274,  0.628408287342, -1.953101936801},
        { 1.004987562112, -0.000000000000,  0.000000000000,  0.000000000000},
        { 0.439914558365,  3.951184960439,  0.566299789962, -5.145062579251},
        { 0.503461901514, -8.197729062929,  0.503006506962,  5.864754182371},
        { 0.503710502314, -9.358296801935,  0.503710502314,  9.358296801935},
        { 0.566299789962, -5.145062579251,  0.439914558365,  3.951184960440},
        { 0.625292414313,  1.992340756031,  0.375056870759, -3.550981208131},
        { 1.000610097194,  0.598404825001,  0.000000000000,  0.000000000000},
        { 0.500305048597,  4.787238600005,  0.500305048597, -4.787238600005},
        { 0.503461901514,  8.197729062929,  0.503006506962, -5.864754182371},
        { 0.566299789962,  5.145062579251,  0.439914558365, -3.951184960440},
        { 0.503710502314,  9.358296801935,  0.503710502314, -9.358296801935},
        { 0.376926970544, -3.481012896274,  0.628408287342,  1.953101936801},
        { 0.439914558365, -3.951184960439,  0.566299789962,  5.145062579251},
        { 1.004987562112,  0.000000000000,  0.000000000000,  0.000000000000}
};

/**************************************************************
 *
 *     xy.def - xy-plane curve descriptors for 'lpp.c'
 *
 **************************************************************/
struct DESCRIPTOR{
      double l1,A1,l2,A2;
};

struct DESCRIPTOR desc[43] = {
    {  1.000000000000,   0.000000000000},
    {  0.426861119501, -20.677652279190,   0.600965473691,  20.227029553329},
    {  0.426861119501,  20.677652279190,   0.600965473691, -20.227029553329},
    {  0.000000000000,   0.000000000000},
    {  1.450450440178,   1.265295220167},
    {  0.475286674856,  -6.072725411590,   1.082989700434,   5.743097253612},
    {  0.832934089063,   5.272056359189,   0.600005018599,  -5.166871437198},
```

```
    {   0.000000000000,   0.000000000000},
    {   1.450450440178,  -1.265295220167},
    {   0.832934089063,  -5.272056359189,   0.600005018599,   5.166871437198},
    {   0.475286674856,   6.072725411590,   1.082989700434,  -5.743097253612},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.600965473691,  20.227029553329,   0.426861119501, -20.677652279190},
    {   1.054318417319,   4.747378858590},
    {   0.527159208659,  37.979030868724,   0.527159208659, -37.979030868724},
    {   0.000000000000,   0.000000000000},
    {   1.082989700434,   5.743097253612,   0.475286674856,  -6.072725411590},
    {   1.652500089655,   2.088558538488},
    {   0.902131678928,  10.078999384402,   0.628250853818, -10.838140077661},
    {   0.000000000000,   0.000000000000},
    {   0.600005018599,   5.166871437198,   0.832934089063,  -5.272056359189},
    {   1.414213562373,   0.000000000000},
    {   0.628250853818,  10.838140077661,   0.902131678928, -10.078999384402},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   0.600965473691, -20.227029553329,   0.426861119501,  20.677652279190},
    {   0.527159208659, -37.979030868724,   0.527159208659,  37.979030868724},
    {   1.054318417319,  -4.747378858590},
    {   0.000000000000,   0.000000000000},
    {   0.600005018599,  -5.166871437198,   0.832934089063,   5.272056359189},
    {   0.628250853818, -10.838140077661,   0.902131678928,  10.078999384402},
    {   1.414213562373,  -0.000000000000},
    {   0.000000000000,   0.000000000000},
    {   1.082989700434,  -5.743097253612,   0.475286674856,   6.072725411590},
    {   0.902131678928, -10.078999384402,   0.628250853818,  10.838140077661},
    {   1.652500089655,  -2.088558538488}
};

/*****************************************************************************
 *
 *   z.def -- z-plane curve descriptors based on 10/70 grid
 *
 *****************************************************************************/
```

```
struct DESCRIPTOR z_desc[43] = {
      { 1.000000000000,  0.000000000000,  0.000000000000,  0.000000000000},
      { 0.375115235242, -5.066081287242,  0.625593247974,  2.839641101518},
      { 0.375115235242,  5.066081287242,  0.625593247974, -2.839641101518},
      { 0.505688790039,  6.550461901757,  0.505688790039, -6.550461901757},
      { 0.631932817268,  2.727297694553,  0.378921326300, -4.865469783722},
      { 0.506118022287,  8.183874149810,  0.507051887040,-11.420667239510},
      { 0.505688790039, -6.550461901757,  0.505688790039,  6.550461901757},
      { 0.506118022287, -8.183874149810,  0.507051887040, 11.420667239510},
      { 0.631932817268, -2.727297694553,  0.378921326300,  4.865469783722},
      { 0.625593247974, -2.839641101518,  0.375115235242,  5.066081287241},
      { 0.500619316543, -6.820142254254,  0.500619316543,  6.820142254254},
      { 1.001238633086, -0.852517781782,  0.000000000000,  0.000000000000},
      { 0.378921326300,  4.865469783722,  0.631932817268, -2.727297694553},
      { 1.010152544552,  0.000000000000,  0.000000000000,  0.000000000000},
      { 0.442414807789,  5.530421017324,  0.570239645211, -7.174182722803},
      { 0.507051887040,-11.420667239510,  0.506118022287,  8.183874149809},
      { 0.507556871107,-13.022683702248,  0.507556871107, 13.022683702248},
      { 0.570239645211, -7.174182722804,  0.442414807789,  5.530421017324},
      { 0.625593247974,  2.839641101518,  0.375115235242, -5.066081287241},
      { 1.001238633086,  0.852517781782,  0.000000000000,  0.000000000000},
      { 0.500619316543,  6.820142254254,  0.500619316543, -6.820142254254},
      { 0.507051887040, 11.420667239510,  0.506118022287, -8.183874149809},
      { 0.570239645211,  7.174182722804,  0.442414807789, -5.530421017324},
      { 0.507556871107, 13.022683702248,  0.507556871107,-13.022683702248},
      { 0.378921326300, -4.865469783722,  0.631932817268,  2.727297694553},
      { 0.442414807789, -5.530421017324,  0.570239645211,  7.174182722803},
      { 1.010152544552, -0.000000000000,  0.000000000000,  0.000000000000}
};

/*************************************************************************
 *
 *      guid_xyz.c -  This program is the three dimensional guidance
 *                    system for AUV II.
 *
 *      LT M.J. CLOUTIER
 *      04/17/90
 *
 *      Inputs  - file (argv[1]) of waypoints consisting of x,y position
 *
 *      Outputs - file (argv[2]) of input descriptors for stepper
 *              - file (argv[3]) of stepper output postures
 *
```

```
*      Rev7   - 4/16/90 - added third dimension to lpp
*      Rev6   - 4/16/90 - attached another date to code
*      Rev5   - 1/24/89 - modified code to operate with IRIS graphics
*                         routines
*      Rev4   - 1/11/90 - changed get_init_data to return (x0,y0,card_hdg)
*      Rev3   - 1/10/90 - changed dir_chg to operate on sign of norm(hdg)
*                       - changed get_descriptor to tbl lookup (vs switch)
*                       - default cardinal_hdg is now determined from first
*                         two way-pts
*                       - eliminated atan2 using tbl lookup
*                       - shifted output coord system [abs(HPI-theta)]
*      Rev2   - 1/09/90 - incorporated GRID_SZ
*      Rev1   - 1/07/90 - initialized ref_theta, init_hdg
*      Orig   - 12/21/89
*
*
*************************************************************************/

#include <stdio.h>
#include <math.h>
#include "lpp.h"
#include "xy.def"
#include "z.def"

double act_x, act_y, act_z, act_v, act_hdg;
#define act_omega 0.0

extern struct DESCRIPTOR;                    /* (from 'curves.def') */

/* exported GLOBALS to IRIS/SYMBOLICS */
double des_x, des_y, des_z, des_v, des_hdg, des_phi; /* values to AUTOPILOT */

/* globals */
int x_last,y_last,z_last,v_last;    /* holds next goal values from MP */
int x_next,y_next,z_next;           /* current goal position from MP */
int v_next;                         /* next desired velocity from MP */
int v_curr;                         /* current velocity */
int triple;                         /* the current turn combo (eg. LLL) */
int z_triple;
double cardinal_hdg;                /* vehicle cardinal heading (N,S,E,W) */
struct DESCRIPTOR cd;               /* descriptor for current spiral */
struct DESCRIPTOR zd;
double d_trvl;                      /* distance traveled on current spiral */
```

```
double z_trvl;
double st_x, st_y, st_z, st_hdg;    /* values from stepper */
double z_hdg,st_xz;                 /* dummy for curve trvl calc */
FILE *infp, *outfp;                 /* input path, output posture files */
int eof_infp;                      /* eof pointer for input */
double x_start, y_start, z_start;   /* initial position */
int curve_type;                    /* (NO_CRV,SNGL,DBL1,DBL2) */
int z_ctype;

/*************************************************************
 *   MISCELLANEOUS MATH FUNCTIONS
 *************************************************************/

/*************************************************************
 *
 *      ATAN2 - returns table lookup of angle to replace
 *              atan2
 *
 *************************************************************/

/* TABLE LOOKUP VALUES */
double xy_ang[5][5] = {
   -2.356194490192344840,
   -2.034443935795702710,
   -1.570796326794896560,
   -1.107148717794090410,
   -0.785398163397448279,
   -2.677945044588986970,
   -2.356194490192344840,
   -1.570796326794896560,
   -0.785398163397448279,
   -0.463647609000806094,
    3.141592653589793120,
    3.141592653589793120,
    0.000000000000000000,
    0.000000000000000000,
    0.000000000000000000,
    2.677945044588986970,
    2.356194490192344840,
    1.570796326794896560,
    0.785398163397448279,
    0.463647609000806094,
    2.356194490192344840,
```

```
    2.034443935795702710,
    1.570796326794896560,
    1.107148717794090410,
    0.785398163397448279};

double ATAN2(y2,y1,x2,x1)
int y2,y1,x2,x1;
{
    return xy_ang[(y2-y1)/GRID_SZ+2][(x2-x1)/GRID_SZ+2];
} /* ATAN2 */

/*************************************************************
 *
 *      norm - normalizes angle between -PI and PI
 *
 *************************************************************/
double norm(a)
double a;
{
    while (a > PI_)
      a -= DPI;
    while (a < -PI_)
      a += DPI;
    return (a);
} /* norm */

/*************************************************************
 *
 *      dir_chg - returns the relative direction chg (L,C or R)
 *                based on input waypoints. dir_chg uses the
 *                global "cardinal_hdg" to determine rel chg
 *
 *************************************************************/
int dir_chg(x1,y1,x2,y2)
int x1,y1,x2,y2;
{
    double heading;
    int rel_dir;

    heading = ATAN2(y2,y1,x2,x1);
    heading -= cardinal_hdg;
    rel_dir = (int)(10.0*norm(heading));
    switch (rel_dir){
```

```
        case LFT    : cardinal_hdg += HPI;
                      return L;
        case CENTER : return C;
        case RGHT   : cardinal_hdg -= HPI;
                      return R;
        default     : fprintf(stderr,"dir_chg:  Bad result in dir_chg\n");
                      exit(-1);
    }
} /* dir_chg */

/*****************************************************************
 *
 *    z_chg - returns the relative depth direction change (U,S,D)
 *            uses normalized (Z_GRID) depth diff for switch
 *
 *****************************************************************/
int z_chg(z1,z2)
int z1,z2;
{
  switch ((z1 - z2)/Z_GRID){
    case -1:  return U;
    case  0:  return S;
    case  1:  return D;
    default:  fprintf(stderr,"z_chg: Bad result\n");
              exit(-1);
  }
} /* z_chg */

/*************************************************************
 *
 *   process_next_waypt - reads next waypt from infp and returns
 *                        the next curve descriptor
 *
 *************************************************************/
void process_next_waypt()
{
   int mp_x, mp_y, mp_z, mp_v;     /* values from MISSION PLANNER */

   if ((eof_infp = fscanf(infp,"%d %d %d %d",&mp_x,&mp_y,&mp_z,&mp_v)) !=
EOF){
       triple = (triple & 0x0f) << 2 | dir_chg(x_last,y_last,mp_x,mp_y);
       z_triple = (z_triple & 0x0f) << 2 | z_chg(z_last, mp_z);
       cd = desc[triple];
```

```
        zd = z_desc[z_triple];
        v_curr = v_next; /* this is target velocity */

        x_next = x_last; /* this is current goal */
        y_next = y_last;
        z_next = z_last;
        v_next = v_last;

        x_last = mp_x;   /* this is next goal */
        y_last = mp_y;
        z_last = mp_z;
        v_last = mp_v;
    }
} /* process_next_waypt */

/*************************************************************
 *      next_pos - calculates next reference values for
 *                 x, y and theta
 *************************************************************/
void next_pos(del_s,A,l)
double del_s,A,l;
{
    double theta, del_theta, t_step;
    double kappa;

/* $$$$$$$$ DEBUGGING VALUES $$$$$$$$ */
act_x = st_x;
act_y = st_y;
act_hdg = st_hdg;
/* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ */

    d_trvl += del_s;

    kappa = A * d_trvl * (l - d_trvl);
    del_theta = kappa * del_s;
    st_hdg += del_theta;
    theta = st_hdg - del_theta/2.0;
    if (abs(del_theta) == 0.0){
        t_step = del_s;
    } else {
        t_step = del_s * (sin(del_theta / 2.0)/(del_theta / 2.0));
    }
    st_x += cos(theta) * t_step;
```

```
    st_y += sin(theta) * t_step;
} /* next_pos */

/*************************************************************
 *      next_z - calculates next reference values for z
 *************************************************************/
void next_z(del_sz,A,l)
double del_sz,A,l;
{
    double phi, del_phi, t_sz;
    double kappa_z;

/************************* DEBUGGING ************/
    act_z = st_z;

    z_trvl += del_sz;

    kappa_z = A * z_trvl * (l - d_trvl);
    del_phi = kappa_z * del_sz;
    z_hdg += del_phi;
    phi = z_hdg - del_phi/2.0;
    if (abs(del_phi) == 0.0){
        t_sz = del_sz;
    } else {
        t_sz = del_sz * (sin(del_phi / 2.0)/(del_phi / 2.0));
    }
    st_xz += cos(phi) * t_sz;
    st_z += sin(phi) * t_sz;
} /* next_pos */

/*************************************************************
 *      print_stepper_out - prints x,y and theta to step_outfp
 *************************************************************/
void print_stepper_out()
{
    fprintf(outfp,"%d %d %8.4f\n",
        st_x, st_y, norm(fabs(HPI - st_hdg)));
}

/*************************************************************
 *      print_desired_out - prints x,y and theta to step_outfp
 *************************************************************/
void print_desired_out()
```

```
{
   fprintf(outfp,"%8.4f %8.4f\n", des_x + x_start, des_y + y_start);
}

/*************************************************************
 *      step_spiral - breaks spiral into discrete postures,
 *                    prints to output file with break between
 *                    curves
 *************************************************************/
void step_spiral()
{
  int xy_ct;
  int z_ct;
  double step;
  double z_step;

  xy_ct = 0;
  z_ct = 0;
  while(xy_ct < MAX_CT){
    step = des_v * RPM_2_VEL * INTRVL;
    z_step = step * (cd.l1 + cd.l2) / (zd.l1 + zd.l2);
    switch(curve_type){
        case NO_CRV: d_trvl = 0.0;
                     /* get spiral type */
                     if (cd.l2 == 0.0){
                   curve_type = SNGL;
                  } else {
                   curve_type = DBL1;
                  }
                  /* step to next_pos */
                  for (;((d_trvl <= cd.l1) && (xy_ct < MAX_CT)); xy_ct++){
                   next_pos(step,cd.A1,cd.l1);
                  }
                  break;
      case SNGL:
      case DBL1:   for (;((d_trvl <= cd.l1) && (xy_ct < MAX_CT)); xy_ct++){
                   next_pos(step,cd.A1,cd.l1);
                  }
                  /* finished this spiral, reset d_trvled, do next */
                  if (d_trvl >= cd.l1){
                   d_trvl = 0.0;
                   /* if second part exists => start along it */
                   if (cd.l2 != 0.0){
```

```
                curve_type = DBL2;
                for (;((d_trvl <= cd.l2) && (xy_ct < MAX_CT)); xy_ct++){
                  next_pos(step,cd.A2,cd.l2);
                }
              } else { /* go to next way_pt */
                curve_type = NO_CRV;
              }
            }
            break;
  case DBL2:  for (;((d_trvl <= cd.l2) && (xy_ct < MAX_CT)); xy_ct++){
                next_pos(step,cd.A2,cd.l2);
              }
              if (d_trvl >= cd.l2){
                d_trvl = 0.0;
                curve_type = NO_CRV;
              }
              break;
    default:  fprintf(stderr,"step_spiral:  Bad input value\n");
}
switch (z_ctype){
  case NO_CRV: z_trvl = 0.0;
              /* get spiral type */
              if (curve_type != NO_CRV){ /* started a curve */
                if (zd.l2 == 0.0){
                  z_ctype = SNGL;
                } else {
                  z_ctype = DBL1;
                }
                /* step to next_pos */
                for (;((z_trvl <= zd.l1) && (z_ct < MAX_CT)); z_ct++){
                  next_z(z_step,zd.A1,zd.l1);
                }
              }
              break;
      case SNGL:
   case DBL1:  for (;((z_trvl <= zd.l1) && (z_ct < MAX_CT)); z_ct++){
                next_z(z_step,zd.A1,zd.l1);
              }
              /* finished this spiral, reset z_trvled, do next */
              if (z_trvl >= zd.l1){
                z_trvl = 0.0;
                /* if second part exists => start along it */
                if (zd.l2 != 0.0){
```

```
                    z_ctype = DBL2;
                    for (;((z_trvl <= zd.l2) && (z_ct < MAX_CT)); z_ct++){
                      next_z(z_step,zd.A2,zd.l2);
                    }
                  } else { /* go to next way_pt */
                    z_ctype = NO_CRV;
                  }
                }
                break;
      case DBL2:  for (;((z_trvl <= zd.l2) && (z_ct < MAX_CT));z_ct++){
                    next_z(z_step,zd.A2,zd.l2);
                  }
                  if (z_trvl >= zd.l2){
                    z_trvl = 0.0;
                             z_ctype = NO_CRV;
                  }
                    break;
       default:   fprintf(stderr,"step_spiral:  Bad input value\n");
    }
    if (curve_type == NO_CRV){
      process_next_waypt();
    }
  }
printf(" lpp (x,y,z) (%4.3f,%4.3f,%4.3f) MP (x,y,z) (%d,%d,%d)\n",
    st_x*(double)GRID_SZ+x_start,st_y*(double)GRID_SZ+y_start,
    st_z*(double)Z_GRID+z_start,x_next,y_next, z_next);
    print_desired_out();
} /* step_spiral */

/*************************************************************
 *
 *      calc_des_val() - finds error values and desired values
 *                       to feed to AUTOPILOT
 *
 *************************************************************/
void calc_des_val()
{
  double err_x, err_y, err_z, err_hdg;

  err_x = (((st_x - act_x)*cos(act_hdg)) + ((st_y - act_y)*sin(act_hdg)))/2.57;
  err_y = (((act_x - st_x)*sin(act_hdg)) + ((st_y - act_y)*cos(act_hdg)))/2.57;
  err_z = (((st_z - act_z)*cos(act_hdg)) + ((st_z - act_z)*sin(act_hdg)))/2.57;
  err_hdg = norm(st_hdg - act_hdg);
```

```
  des_x = (st_x + err_x) * (double)GRID_SZ;
  des_y = (st_y + err_y) * (double)GRID_SZ;
  des_z = (st_z + err_z) * (double)Z_GRID;
  des_v = (K_X * err_x) + ((double)v_curr * cos(err_hdg));
  des_hdg += INTRVL*(act_omega + (v_curr * (K_Y * err_y + K_HDG *
err_hdg)));
/*************TALK TO PROF KANAYAMA ********/
/*    des_phi += INTRVL*()*/
  /* if not too far behind, then get next st_ values else use old ones */
  if (err_x <= MIN_X_ERROR){
     step_spiral();
  }
} /* calc_des_val */

/*************************************************************
 *
 *      open_files() - open input & output files
 *
 *************************************************************/
void open_files()
{
   if ((infp = fopen("path.in","r")) == 0) {
      printf("Unable to open 'path.in'\n");
      exit(-1);
   }
   if ((outfp = fopen("path.out","w")) == 0) {
      printf("Unable to open 'path.out'\n");
      exit(-1);
   }
} /* open_files */

/*************************************************************
 *
 *      get_init_values - get initial lpp values
 *
 *************************************************************/
void get_init_values()
{
   int dx1,dy1,dx2,dy2;
   int dir1,dir2,dir3;
   int x_curr, y_curr, z_curr;
   int x_past, y_past, z_past;
```

```
/* initialize variables */
x_past = y_past = 0;
x_curr = y_curr = z_curr = v_curr = 0;
x_next = y_next = z_next = v_next = 0;
x_last = y_last = z_last = v_last = 0;

/* this is the first time through, so read four waypoints */
if ((eof_infp = fscanf(infp,"%d %d %d %d   %d %d %d %d   %d %d %d %d",
    &x_curr,&y_curr,&z_curr,&v_curr,
    &x_next,&y_next,&z_next,&v_next,
    &x_last,&y_last,&z_last,&v_last)) != EOF) {

    /* get initial data */
    dx1 = SIGN(x_next - x_curr);
    dy1 = SIGN(y_next - y_curr);
    dx2 = SIGN(x_last - x_next);
    dy2 = SIGN(y_last - y_next);
    x_start = x_curr;
    y_start = y_curr;
    z_start = z_curr;

    /* this switch determines a valid position PRIOR to start */
    switch((dx1 << 2) | dy1){
      case PZ: cardinal_hdg = ZERO;
               x_past = x_curr - GRID_SZ;
               y_past = y_curr;
               break;
      case PP: switch ((dx2 << 2) | dy2){
                 case PN: cardinal_hdg = HPI;
                          x_past = x_curr;
                          y_past = y_curr - GRID_SZ;
                          break;
                 default: cardinal_hdg = ZERO;
                          x_past = x_curr - GRID_SZ;
                          y_past = y_curr;
               }
               break;
      case PN: switch ((dx2 << 2) | dy2){
                 case PP: cardinal_hdg = -HPI;
                          x_past = x_curr;
                          y_past = y_curr + GRID_SZ;
                          break;
                 default: cardinal_hdg = ZERO;
```

```
                x_past = x_curr - GRID_SZ;
                y_past = y_curr;
        }
        break;
  case ZP: cardinal_hdg = HPI;
        x_past = x_curr;
        y_past = y_curr - GRID_SZ;
        break;
  case ZN: cardinal_hdg = -HPI;
        x_past = x_curr;
        y_past = y_curr + GRID_SZ;
        break;
  case NZ: cardinal_hdg = PI_;
        x_past = x_curr + GRID_SZ;
        y_past = y_curr;
        break;
  case NP: switch ((dx2 << 2) | dy2){
          case NN: cardinal_hdg = HPI;
                x_past = x_curr;
                y_past = y_curr - GRID_SZ;
                break;
          default: cardinal_hdg = PI_;
                x_past = x_curr + GRID_SZ;
                y_past = y_curr;
        }
        break;
  case NN: switch ((dx2 << 2) | dy2){
          case NP: cardinal_hdg = -HPI;
                x_past = x_curr;
                y_past = y_curr + GRID_SZ;
                break;
          default: cardinal_hdg = PI_;
                x_past = x_curr + GRID_SZ;
                y_past = y_curr;
        }
        break;
  default: fprintf(stderr,"get_init_hdg: Bad input\n");
        exit(-1);
}

/* now get triple */
dir1 = dir_chg(x_past,y_past,x_curr,y_curr);
dir2 = dir_chg(x_curr,y_curr,x_next,y_next);
```

```
        dir3 = dir_chg(x_next,y_next,x_last,y_last);
        triple = (dir1 << 4) | (dir2 <<2) | dir3;

        /* and 'current' vehicle heading */
        st_hdg = ATAN2(y_next,y_past,x_next,x_past);

        /* for now the desired depth is MP depth */
        des_hdg = st_hdg;
        des_z = (double) z_next;
        des_v = (double) v_curr;
    } else {
        fprintf(stderr,"lpp_step main: not enough waypoints for path\n");
        exit(-1);
    }
    /* read curve information */
    cd = desc[triple];
    /* and get first desired values */
    step_spiral();
} /* get_init_values */

main()
{
    int setparm = TRUE_;

    while (eof_infp != EOF){
      if (setparm){
        open_files();
        get_init_values();
        setparm = FALSE_;
      }
      calc_des_val();
    }
    close(infp);
    close(outfp);
}
/* guid_xyz.c */
```

LIST OF REFERENCES

[Albus 88]
Albus, J.S., "System Description and Design Architecture for Multiple Autonomous Vehicles," *NIST Technical Note 1251*, September 1988.

[Albus 89]
Albus, J.S., McCain, H.G., and Lumia, R., "NASA/NBS Standard Reference Model for Telerobot Control Systems Architecture," *NIST Technical Note 1235*, April 1989.

[Albus 90]
Albus, J.S., "An Architecture for Real-Time Intelligent Control of Submarines," paper presented at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

[Alcyon 86]
REGULUS - Programmers Reference Manual, Alcyon Corporation, San Diego, CA, August 1986.

[Arkin 87]
Arkin, R.C., "Motor Schema-Based Mobile Robot Navigation," *Proc. of IEEE International Conf. on Robotics and Automation*, pp. 264-271, IEEE Press, New York, April 1987.

[Baker 89]
Baker, A.D. III, "Combat Fleets," U.S. Naval Institute *Proceedings*, Vol. 115, No. 5, p. 158, May 1989.

[Barker 86]
Barker, R., and Tudor-Craig, M., "Concept of an Autonomous Vehicle for Underwater Tasks," *Proc. of Subsea Defense Conference*, Subsea Conferences Ltd., Bristol, England, 1986.

[Barry 88]
Barry, J.M., and Sary, C., "Expert System for On-board Satellite Scheduling and Control," *Proc. of the Fourth Conference on Artificial Intelligence for Space Applications*, NASA Conf. Pub. 3013, pp. 193-203, NASA Scientific and Technical Information Branch, 1988.

[Bellingham 89]
Bellingham, J. and others, "An Autonomous Submersible Designed for Software Development," *Proc. of the MTS/IEEE OCEANS '89 Conference*, pp. 799-803, IEEE Marine Technology Society, Seattle, WA, 1989.

[Bellingham 90]
Bellingham, J., "Applying Layered Controls to AUVs," paper presented at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

[Blidberg 90]
Blidberg, R., "EAVE III," presentation at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

[Brady 82]
Brady, M., "Trajectory Planning and Robot Motion Planning and Control," *MIT Press*, Cambridge, MA, 1982.

[Brooks 88]
Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," *IEEE Jour. of Robotics and Automation*, RA-2, 14, 1986.

[Brooks 89]
Brooks, R.A., and Flynn, A.M., "A Robot Being," unpublished paper from MIT Artificial Intelligence Laboratory, 1989.

[Busby 89]
Busby, R.F., and Vadus, J.R., "Status and Trend in Autonomous Underwater Vehicles (AUV) Research and Development in the U.S.A.," *National Oceanic and Atmospheric Administration*, Rockville, MD, September 1989.

[Brunner 88]
Brunner, G.M., *Experimental Verification of AUV Performance*, Mechanical Engineers Thesis, Naval Postgraduate School, Monterey, CA, March 1988.

[Chappell 87]
Chappell, S.G., "A Blackboard Based System for Context Sensitive Mission Planning in an Autonomous Vehicle," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol II, pp. 467-476, University of New Hampshire, Marine Systems Laboratory, June, 1987.

[Crowley 85]
Crowley, J.L., "Navigation for an Intelligent Mobile Robot," *Proc. of 1985 Conference on Artificial Intelligence Applications*, pp. 79-84, Computer Society Press, Washington D.C., 1985.

[Data 89]
1988/1989 Data Acquisition Handbook, Part III, Section 3, Data Translation, Inc., Marlboro, MA, 1989.

[Dettman 88]
Dettman, T.R., *DOS Programmers Reference*, pp. 310-332, QUE Corp., 1988.

[Dibble 88]
Dibble, P., *OS-9 Insights - An Advanced Programmers Guide to OS9/68000*, Microware Systems, 1988.

[Dougherty 88]
Dougherty, F., and others, "An Autonomous Underwater Vehicle (AUV) Flight Control Using Sliding Mode Control," *Proc. of OCEANS '88*, pp. 1265-1269, IEEE Marine Technology Society, Baltimore, MD, November 1988.

[Durham 87]
Durham, J.T., and others, "EAVE-West: A Testbed for Plan Execution," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol I, pp. 33-44, University of New Hampshire, Marine Systems Laboratory, June, 1987.

[Durham 90]
Durham, J.T., untitled presentation at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

[Friend 89]
Friend, J., *Design of a Navigator for a Testbed Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1989.

[GESPAC 88]
GESMOS-68 OS-9/68000 Operating System, Ver. 2.2, GESPAC, Inc., Geneva, SA, 1988.

[GESPAC 89]
GESPAC 1989 - Boards, Systems, Software, Product Catalog, GESPAC, Inc., Geneva, SA, 1989.

[Good 90]
Good, M., *Design and Construction of the Second Generation AUV*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1989.

[GRiD 88]
GRiD Systems Corp., *GRiDCASE 1535 EXP Owner's Guide (1535-40)*, Rev. A, Fremont, CA, November 1988.

[Healy 89b]
Healy, A.J., and others, "Planning, Navigation, Dynamics and Control of Autonomous Underwater Vehicles," Report of Progress on Direct Fund Research at the Naval Postgraduate School, July 1989.

[Healy 90]
Healy, A.J., and others, "Planning, Navigation, Dynamics and Control of Autonomous Underwater Vehicles," Proposal for Research submitted to NSWC, White Oak Laboratories, Naval Postgraduate School, May 1990.

[Hebert 87]
Hebert, M., and others, "A Feasibility Study for a Long Range Autonomous Underwater Vehicle," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol I, pp. 1-14, University of New Hampshire, Marine Systems Laboratory, June, 1987.

[Herman 88]
Herman, M., and Albus, J.S., "Overview of the Multiple Automous Vehicles (MAUV) Project," *Proc. of IEEE Conf. on Robotics and Automation*, Vol. 1, pp. 618-621, IEEE Press, 1988.

[Holzer 90]
Holzer, R., "AUV's Mission to Grow for Navy," *Defense News*, Vol. 5, No. 15, April 1990.

[Isik 84]
Isik, C., and Meystel, A., "Knowledge-Based Pilot for an Intelligent Mobile Autonomous System," *Proc. of 1984 Conference on Artificial Intelligence Applications*, pp. 57-63, Computer Society, Denver, CO, 1984.

[Jalbert 88]
Jalbert, J. and others, "EAVE III Untethered AUV Submersible," *Proc. of IEEE Oceans '88*, pp. 1259-1263, Baltimore, MD, November, 1988.

[Jackson 84]
Jackson, E., and Ferguson, J., "Design of ARCS - Autonomous Remotely Controlled Submersible," *ROV '84 Conference Proc.*, pp. 365-368, IEEE Marine Technology Society, 1984.

[Kanayama 88a]
Kanayama, Y., and Hartman, B.I., "Smooth Local Path Planning for Autonomous Vehicles," paper submitted to *The International Jour. of Robotics Research*, June 1988.

[Kanayama 88b]
Kanayama, Y., and Hartman, B.I., "Smooth Local Path Planning for Autonomous Vehicles," *Proc. of IEEE International Conf. on Robotics and Automation*, pp. 1265-1270, 1989.

[Kanayama 90]
Kanayama, Y., and others, "A Stable Tracking Control Method for an Autonomous Mobile Robot," *Proc. of IEEE International Conf. on Robotics and Automation*, Cincinnati, OH, May 1990.

[Khatib 85]
Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proc. of IEEE International Conf. on Robotics and Automation*, pp. 500-505, IEEE Press, 1985.

[Kwak 90]
Kwak, S.H., Ong, S.M., and McGhee, R.B., "A Mission Planning Expert System for an Autonomous Underwater Vehicle," *Proc. of IEEE Conf. on Autonomous Underwater Vehicles*, Washington, DC, June 1990.

[Kyriakopoulos 88]
Kyriakopoulos, K.J., "Minimum Jerk Path Generation," *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pp. 364-369, Computer Society Press, Washington D.C.,1988.

[Mayer 87]
Mayer, R., and others, "Situation Based Control Architecture for an AUV," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol II, pp. 430-443, University of New Hampshire, Marine Systems Laboratory, June, 1987.

[Microware 87]
Using Professional OS9, Microware Systems Corp., Des Moines, IA, 1987.

[Mozier 87]
Mozier, A., and Pagurek, B., "An Onboard Navigation System for Autonomous Underwater Vehicles," *Proc. of International Conf. on Intelligent Autonomous Systems*, L.O. Hertzberger, ed., pp. 449-459, Elsivier Science, Amsterdam, 1987.

[Nitao 85]
Nitao, J.J., and Parodi, A.M., "An Intelligent Pilot for an Autonomous Vehicle System," *Proc. of 1985 Conference on Artificial Intelligence Applications*, pp. 176-183, Computer Society Press, Washington D.C., 1985.

[Nitao 86]
Nitao, J.J., and Parodi, A.M., "A Real-Time Reflexive Pilot for an Autonomous Land Vehicle," *IEEE Control Systems Magazine*, pp. 14-23, February 1986.

[Ong 90]
Ong, S.M., *A Mission Planning Expert System with Three-Dimensional Path Optimization for the NPS Model 2 Autonomous Underwater Vehicle*, MS Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

[Parodi 84]
Parodi, A.M., "A Route Planning System for an Autonomous Vehicle," *Proc. of 1984 Conference on Artificial Intelligence Applications*, pp. 51-56, Computer Society, Denver, CO, 1984.

[Pittman 87]
PITMO D-C Servo Motors, Bulletin 14000, Pittman Corp, Harleysville, PA, June 1987.

[Riling 90]
Riling, D., *Real-Time Implementation of an Autopilot and Signal Processor for an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1990.

[Robinson 86]
Robinson, R.C., "National Defense Applications of Autonomous Underwater Vehicles," *IEEE Journal of Ocean Engineering*, Vol. OE-11, No. 4, pp. 462-467, October 1986.

[Russell 86]
Russell, G.T., and Lane, D.M., "A Knowledge-Based System Framework for Enviornmental Perception in a Subsea Robotics Context," *IEEE Jour. of Oceanic Engineering*, Vol. OE-11, No. 3, pp. 401-412, October 1986.

[Shin 86]
Shin, K.G., and McKay, N.D., "Minimum Time Trajectory Planning for Industrial Robots with General Torque Constraints," *Proc. of IEEE International Conf. on Robotics and Automation*, pp. 412-417, IEEE Press, April 1986.

[Thomas 85]
Thomas, B., "Control Software in the ARCS Vehicle," *Proc. of 1985 International Symposium on Unmanned Untethered Submersible Technology*, pp. 334-341, 1985.

[Weitzel 87]
Weitzel, R.G., "DOLPHIN: New Applications and Technology," *Proc. of the MTS ROV '87 Conference*, pp. 256-262, IEEE Marine Technology Society, San Diego, March 1987.

[Yoerger 90]
Yoerger, D.R., "ARGO-JASON Control Systems," presented at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

[Zheng 88]
Zheng, X., and Srivastava, S., "Events and Actions: An Object-Oriented Approach to Real-Time Control Systems," *Proc. of OCEANS '88*, IEEE Marine Technology Society, Baltimore, MD, November 1988.

[Zheng 90]
Zheng, X., "The Design of a User-Configurable Real-Time System," presented at MBARI Software Architectures Meeting, Monterey, CA, February 1990.

BIBLIOGRAPHY

Antsaklis, P.J., Passino, K.M., and Wang, S.J., "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues," paper submitted to *Journal of Intelligent and Robotic Systems*, 1989.

Bakkers, A.P., and Verhoven, W.L.A., "A Robot Control Algorithm Implementation in Transputers," *Proc. of International Conf. on Intelligent Autonomous Systems*, L.O. Hertzberger, ed., pp. 118-128, Elsivier Science, Amsterdam, 1987.

Bane, G.L., and Ferguson, J., "The Evolutionary Development of the Military Autonomous Underwater Vehicle," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol I, pp. 60-88, University of New Hampshire, Marine Systems Laboratory, June, 1987.

Brady, M., Gerhardt, L., and Davidson, H., "Artificial Intelligence and Robotics," *Robotics and Artificial Intelligence, Series F: Computer and Systems Sciences*, Springer-Verlag, Vol.11, p. 47, 1984.

Cyr, R., and Warner, J., "TOAS: Terrain and Obstacle Avoidance Sonar for Autonomous and Remotely Operated Underwater Vehicles," *Proc. of the MTS ROV '87 Conference*, pp. 324-330, San Diego, CA, March 1987.

Davis, M., *Real-Time Adaptive Control for AUV Diving*, Engineer's Thesis, Naval Postgraduate School, Monterey, CA, September 1989.

Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, vol. 26, pp. 251-270, Elsevier Science Publishers, July 1985.

Healy, A.J., Papoulias, F.A., and MacDonald, G., "Design and Experimental Verification of a Model Based Compensator for Rapid AUV Depth Control," *Proc. of the Sixth Unmanned, Untethered, Submermisible Technology Conf*, Washington DC., June 1989.

Herr, W.J., "AUV Technology Development and Demonstration Program," *Proc. of OCEANS '88*, pp. 1290-1299, IEEE Marine Technology Society, Baltimore, MD, November 1988.

Honerd, G., Jongkind, W., and van Aalst, C.H., "Sensor and Navigation System for a Mobile Robot," *Proc. of International Conf. on Intelligent Autonomous Systems*, L.O. Hertzberger, ed., pp. 258-265, Elsivier Science, Amsterdam, 1987.

Ishitani, H., Baba, Y., and Ura, T., "Attitude Control System of a Streamlined Cruising Type Autonomous Submersible," *Proc. of the Sixth International Symposium on Unmanned Untethered Submersible Technology*, University of New Hampshire, Marine Systems Laboratory, 1990.

Iyengar, S.S., and others, "Learned Navigation Paths for a Robot in Unexplored Terrain," *Proc. of 1985 Conference on Artificial Intelligence Applications*, pp. 148-155, Computer Society Press, Washington D.C., 1985.

Jaswa, V.C., *An Experimental Study of Real-Time Computer Control of a Hexapod Vehicle*, Ph.D. Dissertation, Department of Electrical Engineering, The Ohio State University, Columbus, OH, June 1978.

Kanayama, Y. and Noguchi, T., "Locomotion Functions for a Mobile Robot Language," *Proc. of IEEE International Workshop on Intelligent Robots and Systems*, pp. 542-549, September 1989.

Keirsey, D. and others, "Autonomous Vehicle Control Using AI Techiniques," *IEEE Trans. on Software Engineering*, Vol. SE-11, No. 9, pp. 986-990, September 1985.

Kuan, D.T., and others, "Automatic Path Planning for a mobile Robot using a Mixed Representation of Free Space," *Proc. of 1985 Conf. on Artificial Intelligence Applications*, pp. 70-74, Computer Society Press, Washington D.C., 1985.

Laffey, T.J., and others, "Real-Time Knowledge-Based Systems," *AI Magazine*, pp. 27-45, Spring 1988.

Lienard, D., *Sliding Mode Control for Multivariable AUV Autopilots*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

Lock, J.J., and others, "Software Development for the Autonomous Submersible Program at M.I.T. Sea Grant and Draper Laboratory," *Proc. of the Sixth International Symposium on Unmanned Submersible Technology*, University of New Hampshire, Marine Systems Laboratory, 1989.

Maes, P., "The Dynamics of Action Selection," *Proc. of AAAI Spring Symposium on AI Limited Rationality*, pp. 87-91, IJCAI, Detroit, MI, Spring 1989.

MacPherson, D.L., *A Computer Simulation Study of Rule-Based Control of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1988.

Nordman, D.B., *A Computer Simulation Study of Mission Planning and Control for the NPS Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

Pearson, G., and Kuan, D., "Mission Planning System for an Autonomous Vehicle," *Proc. of 1985 Conference on Artificial Intelligence Applications*, pp. 162-1675, Computer Society Press, Washington D.C., 1985.

Pugh, G.E., and Krupp, J.C., "The Control of Autonomous Underwater Vehicles through a Heirarchical Structure of Value Priorities," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol II, pp. 477-501, University of New Hampshire, Marine Systems Laboratory, June 1987.

Rogers, R.C., *A Study of 3-D Visualization and Knowledge-Based Mission Planning and Control for the NPS Model 2 Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1989.

Russo, V., Johnston, G., Campbell, R., "Process Management and Exception Handling in Multiprocessor Operating Systems using Object-Oriented Design Techniques," *OOPSLA '88 Proceedings*, pp. 248-258, September 1988.

Shevenell, M.P., "Hardware and Software Architectures for Realizing a Knowledge Based System on EAVE," *Proc. of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, Vol I, pp. 220-237, University of New Hampshire, Marine Systems Laboratory, June 1987.

Soetadji, T., "Cube Based Representation of Free Space for the Navigation of an Autonomous Mobile Robot," *Proc. of International Conf. on Intelligent Autonomous Systems*, L.O. Hertzberger, ed., pp. 449-459, Elsivier Science, Amsterdam, 1987.

Steele, S., "Remotely Piloted Vehicles," U.S. Naval Institute *Proceedings*, Vol. 114, No. 6, p. 78, June 1988.

Saunders, T.E., *Performance of Small Thrusters and Propulsion Systems*, Masters Thesis, Naval Postgraduate School, Monterey, CA, March 1990.

Tuijnman, F., and others, "A Model for Control Software and Sensor Algorithms for an Autonomous Mobile Robot," *Proc. of International Conf. on Intelligent Autonomous Systems*, L.O. Hertzberger, ed., pp. 449-459, Elsivier Science, Amsterdam, 1987.

Wilfong, G.T., "Motion Planning for an Autonomous Vehicle," *Proc. of 1988 IEEE Conf. on Robotics and Automation*, pp. 529-533, Computer Society Press, Washington D.C.,1988.

Yoerger, D.R., Newman, J.B, and Slotine, J.J.E., "Supervisory Control for the JASON ROV," *IEEE Journal of Ocean Engineering*, Vol. OE-11, No. 3, pp. 392-399, July 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate Schoool Monterey, CA 93943-5002	2
3.	Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5002	1
4.	Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5002 ATTN: Professor Y. Lee, Code CS/Le	30
5.	Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5002 ATTN: LCDR J. Yurchak, Code CS/Yu	1
6.	Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5002 ATTN: Professor Y. Kanayama, Code CS/Ka	1
7.	Chairman, Code 69 Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, CA 93943-5002	2
8.	Space Systems Curricular Officer, Code 39 Naval Postgraduate School Monterey, CA 93943-5002	1

9.	Commander, Naval Surface Warfare Center White Oak, MD 20910 ATTN: H. Cook, Code U25	1
10.	Head, Undersea AI and Robotics Branch Naval Ocean Systems Center San Diego, CA 92152 ATTN: P. Heckman, Code 943	1
11.	Commander, Naval Coastal Systems Center Panama City, FL 32407-5000 ATTN: Dr. G. Dobeck, Code 4210	1
12.	RADM Evans, SEA-92R Naval Sea Systems Command Washington, DC 20362-5101 ATTN: Ms. Judy Rumsey	1
13.	Head, Systems Engineering Branch David Taylor Naval Research and Development Center Carderock Laboratory Bethesda, MD 20084-5000	1
14.	Naval Research Laboratory Marine Systems Division Washington, DC 20032 ATTN: Dr. D. Steiger	1
15.	MIT - Sea Grant College Program 292 Main Street, E38-376 Cambridge, MA 02139 ATTN: Jim Bellingham	1
16.	Naval Plant Representative Office (SSPO) Westinghouse Electric Corporation Hendy Avenue Sunnyvale, CA 94088-3499 ATTN: LT M.J. Cloutier	2
17.	Commander, Naval Surface Warfare Center Silver Springs, MD 20903-5000 ATTN: Jennifer Rau, Code U25	1